CS 342

Fall 2014

Database Project

Orchards Harvest Database

I - Conceptual Database Design and ER model

II - Relational Model and Relational Algebra/Calculus

III - Normalization of Relations

IV - Common Features of PL/SQL and T-SQL

V - Graphics User Interface & Design and Implementation

Team Members

Jason Chi - CS Major

Moises Ayala - CS Major

Database System Project

Table of Contents

Phase 1.....4

1. Fact-Finding Techniques and Information Gathering.....4

1.1 Introduction to Enterprise/Organization.....4

1.2 Description Fact-Finding Techniques.....4

1.3 Project and Database Scope/1.4 Entity sets and relationship sets.....5

1.5 User Groups, Data views and Operations.....6

2.Conceptual Database Design.....7

2.1 Entity Set Description.....7

2.2 Relationship Set Description.....15

Phase 2.....19

3. ER-Model vs. Relational Model.....19

3.1 Description of E-R model and relational model.....19

3.2 Comparisons.....21

3.3 Converting Entity Types to Relations.....21

3.4 Converting Relationship Types to Relations.....22

3.5 Database Constraints.....23

4 Relational Model.....24

4.1 Relation Schema.....24

4.2 Sample Data.....26

4.3 Queries.....30

Phase 3.....36

5. Normalization of Relations.....36

5.1 What is normalization?.....36

5.3 Check Your Relations.....38

5.4 What is SQL * PLUS?.....39

5.5 Oracle/Schema Objects.....40

5.6 Relation Schema.....44

5.7 SQL Queries.....56

Phase 4.....64

6.1 Common Features of PL/SQL and T-SQL.....64

6.2 Purpose of Stored Subprograms.....65

6.3 Benefits of calling stored subprogram over sending a dynamic SQL.....66

6.4 Oracle PL/SQl.....66

6.5 Oracle PL/SQL Subprogram.....73

Phase 5.....75

7.1 Description of Daily activities.....75

7.2 Relation.....76

7.3 Screenshots.....81

7.4 Major steps of designing a user interface.....85

7.5 Descriptions of major classes.....86

7.6 Learning a new development tool in a new language.....88

7.7 Major Steps of Designing and Implementing a Database. .89

Phase 1

1. Fact-Finding Techniques and Information Gathering

1.1 Introduction to Enterprise/Organization

We have taken the task of developing a database for an agriculture automation and analytics company. The software will be deploy at farms to record various data. The purpose of the software is to allow the farm owners to be able to accurately see and follow various set of data. The current method that the farms employ to gather data is not accurate due to it being based on estimation which are often very different from the actual value. This will cause various problems. For example, the farm will estimate how much fruit that they will harvest in a day and ask the packing house to prepare for the the fruit. However, if the estimate is too low then the packing house has not sufficiently prepare for all the fruit. This database will allow the farms to view accurate data and generate accurate reports based on that data.

1.2 Description Fact-Finding Techniques

Our database will be designed to to be able to print out reports on any employee and keep track on the amount of cherries that the employee picks over time. Our database will also keep track of the coordinates location of the bin which will allow the

user to generate yield maps. Yield maps are a graphical overlay on a map which shows the area of the field which are the most productive.

Data will be sent from the fields to our database in a text based file which the extracted data will insert into the database. The data will contain the amount fruit that each employee deposited into the bin, employee identification number, and a time stamp. This data can be delivered either wireless over the internet or it will be delivered using a usb drive. How the data is delivered will depend on whether or not the field where the bin is located at has wifi.

As part of data gathering, we have been talking to many ranch owners, and also have talked to employees that have experience with crops in order to gain knowledge of how the whole harvesting system functions.

1.3 Project and Database Scope/1.4 Entity sets and relationship sets

This database will only generate reports for the specified field of cherries. Usually ranches will have more than one field/orchard of cherries and also they will contain more than one variety of cherries, for example, one field might contain bright sour red cherries, while the other one might contain bing cherries which are two different kinds of cherries. Therefore, our database will keep records for the field specified. We have came up with five major entities which are, Farm, Employee, Fields, Bin, and Crops.

We have a relationship between the Farm entity and the Fields entity which we call “Owns” because a farm owns fields (Pieces of land, usually about 155 acres). We have a relationship from fields to crops named “grows” because fields grow one type and variety of crops. Farm is also related to employee, we call the relationship “Hires” because a farm hires the employee to do all the labor work around the lands. Employee has a relationship with the Crops entity set named “Picks” because employees pick the crops that are grown. The Bin entity set is related to the “Picks” relationship making this a ternary relationship, whenever an employee picks up crops it dumps them into an huge bin, therefore we managed to make bin its own entity set and relate it to the relationship.

1.5 User Groups, Data views and Operations

The groups that will use our database will be the managers, or farm owners whenever they want to know know the coordinates of the bin, in order to generate the maps that will show the trees in the field that gave the most cherries. They would want to know them in order so that they can analyze how good is the soil they are planting in. People in the farm offices will also use our database to generate the reports of the employees number of buckets picked and the amount of weight picked, that way there is solid proof of the work done by the employee.

2.Conceptual Database Design

2.1 Entity Set Description

Farm

The farm entity is to distinguish who is its respective owner, where is the farm located and also what is the name of the farm. This entity has three attributes which will be described below.

Attribute: Location

Description: This is the address of the Ranch, i.e. City, County, and State, where the ranch is located at.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Character String	A-Z 0-9 (100 Characters long)	. .	Yes	Yes	Single	Simple

Attribute: Owner

Description: The name of the owner of the farm.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Character String	A-Z (30 characters long)	..	Yes	Yes	Single	Simple

Attribute: Name

Description: This is the name of the farm/ranch.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Character String	A-Z 0-9 (30 characters long)	..	Yes	Yes	Single	Simple

Fields

Fields are also referred to as orchards, and these are huge pieces of land usually averaging 155 acres, where the farm owner and his staff will plant one type of crop per field. Farms usually contain many fields.

Attribute: Field_id

Description: Farmers usually have a map of their entire farm where they distinguish each field with its respective field ID.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Int	0-10,000	..	No	Yes	Single	Simple

Attribute: Crop

Description: As mentioned before, each field grows different types of crops therefore we need to know what type of crop we are working with.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple	Simple or Composite

					Value	
String	A-Z (35 characters long)	..	Yes	No	Single	Simple

Attribute: Type

Description: Each crop has a different type of variety, for example, when a field grows grapes, there are different types of grapes. You have the big ones with seeds, and then you have green grapes, and purple grapes.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
String	A-Z (35 characters long)	..	Yes	No	Single	Simple

Coordinates

Attribute: Field_Id

This number will be used to identify the field for whom the longtitude and latitude corresponds to.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
int	0-99999	..	No	Yes	Single	Simple

Attribute: Longitude

Description: This the Longitude coordinate of point related to the field. It is a component of Coordinates.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Int	0-10,000	..	Yes	No	Single	Simple

Attribute: Latitude

Description: This the Latitude coordinate of point related to the field. It is a component of Coordinates.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Int	0-10,000	..	Yes	No	Single	Simple

Employee

Attribute: Employee_Id

Description: Every employee has their unique id to distinguish them from the rest of the employee.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Int	0-10,000	..	No	Yes	Single	Simple

Attribute: Name

Description: This stores the employees name.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
String	A-Z (35	..	No	Yes	Single	Simple

	characters long)					

Attribute: Phone

Description: This is used to store the phone number.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Int	0-10,000	..	Yes	Yes	Single	Simple

Attribute: Wage

Description: Wage will represent the amount of money that the employee earns per pound of cherries picked.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Float	0-100.00	..	No	No	Single	Simple

Attribute: Address

Description: This is the address of the Employee.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
String	A-Z 0-9 (100 characters long)	..	No	No	Single	Simple

Bin

Attribute: Bin_id

Description: This is the id of the bin. Usually kept in the records to know where the bin was located at.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Int	0-10,000	..	No	Yes	Single	Simple

Attribute: Weight

Description: This is used to record the amount of weight the bin is holding every time it is updated.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Int	0-10,000	..	No	No	Single	Simple

Attribute: Type

Description: There are different types of bins. Therefore this attribute is used to save the type of bin we are dealing with.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
String	A-Z 0-9 (100 characters long)	..	No	No	Single	Simple

Attribute: Longitude

Description: This the Longitude coordinate of point related to the field. It is a component of Coordinates.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Int	0-10,000	..	Yes	No	Single	Simple

Attribute: Latitude

Description: This the Latitude coordinate of point related to the field. It is a component of Coordinates.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Int	0-10,000	..	Yes	No	Single	Simple

Attribute: Max_Cap

Description: This is the maximum limit of weight of cherries that fit inside the bin. Bins will vary in their max_cap.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Int	0-10,000	..	No	Yes	Single	Simple

Chemical

Attribute: Type

Description: This the type of the chemical being applied to the field. This will include pesticides and fertilizers.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
String	A-Z (35 characters long)	..	No	Yes	Single	Simple

Attribute: Field_Id

This will be used as a foreign key and the parent will be Fields.

Domain/Type	value-range	default value	null value?	unique	single or multiple vlue	simple or composite
int	0-99999	..	no	no	single	simple

Attribute: Chem_id

This will be the id of the particular chemical being applied to the field.

Domain/Type	value-range	default value	null value?	unique	single or multiple vlue	simple or composite
int	0-99999	..	no	no	single	simple

Attribute of Applied Relation: Util_date

This will be the date of when the chemical was applied to the field.

Domain/Type	value-range	default value	null value?	unique	single or multiple	simple or composite

					vlue	
Date	mm/dd/yyyy	..	no	no	single	simple

2.2 Relationship Set Description

Relationship: Owns

Description: This is the relationship between Farm and Fields, meaning that the farm owns various fields.

Entity Sets Involved	Mapping Cardinality	Descriptive Field	Participation Constraint
Farm & Field	1:M	Knowing how many fields are in the Farm	Total

Relationship: Hires

Description: This is the relationship between Farm and Employees, meaning that a farm hires employees. This relationship contains attributes which will be listed below.

Entity Sets Involved	Mapping Cardinality	Descriptive Field	Participation Constraint
Farm & Employee	1:M	Farm hires employees to work on its lands	Total

Attribute of the Hires Relationship: S_date

Attribute of the Hires Relationship: E_date

Description: These attributes are the start date of the employee (S_date) and the end date of the employee (E_date).

Domain/Type	Value-Range	Default Value	Null Values?	Unique	Single or Multiple	Single or Composite

					Value	
Date	Time and Date	..	Yes	No	Single	Simple

Relationship: Picks

Description this is a ternary relationship between Employee, Crops, and Bin. This represents an employee picking up crops and dumping them into a bin.

Entity Sets Involved	Mapping Cardinality	Descriptive Field	Participation Constraint
Employee, Crops, Bin	M:1	Employee picks crops deposits into a bin; therefore, many employees will pick one crop into one bin	Total

Attribute of the Picks Relationship: Date

Description: This attribute is used to know what days and time that the employees deposited up the crops into the bin. This information will be analyzed to create graphs.

Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Date	Date and time	..	Yes	No	Single	Simple

Attribute of the Picks Relationship: Weight

Description: Will be used to determine the amount of cherries picked at any time, can be used to evaluate the employees' working pace.

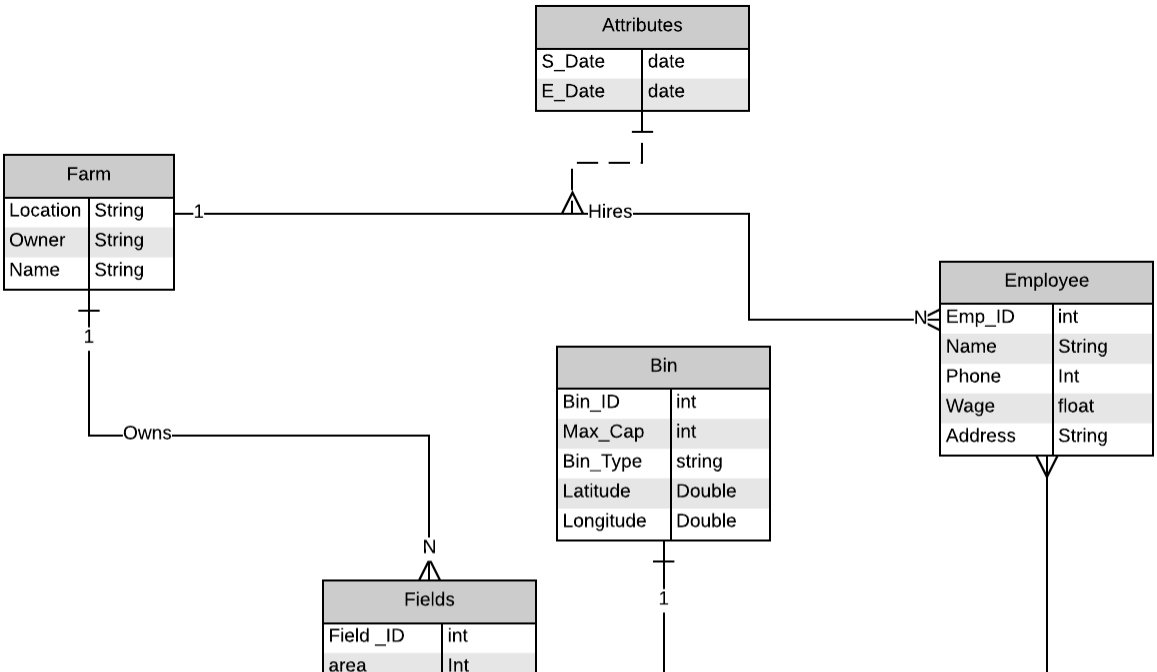
Domain/Type	Value-Range	Default Value	Null Value?	Unique	Single or Multiple Value	Simple or Composite
Int	0-10,000	..	No	Yes	Single	Simple

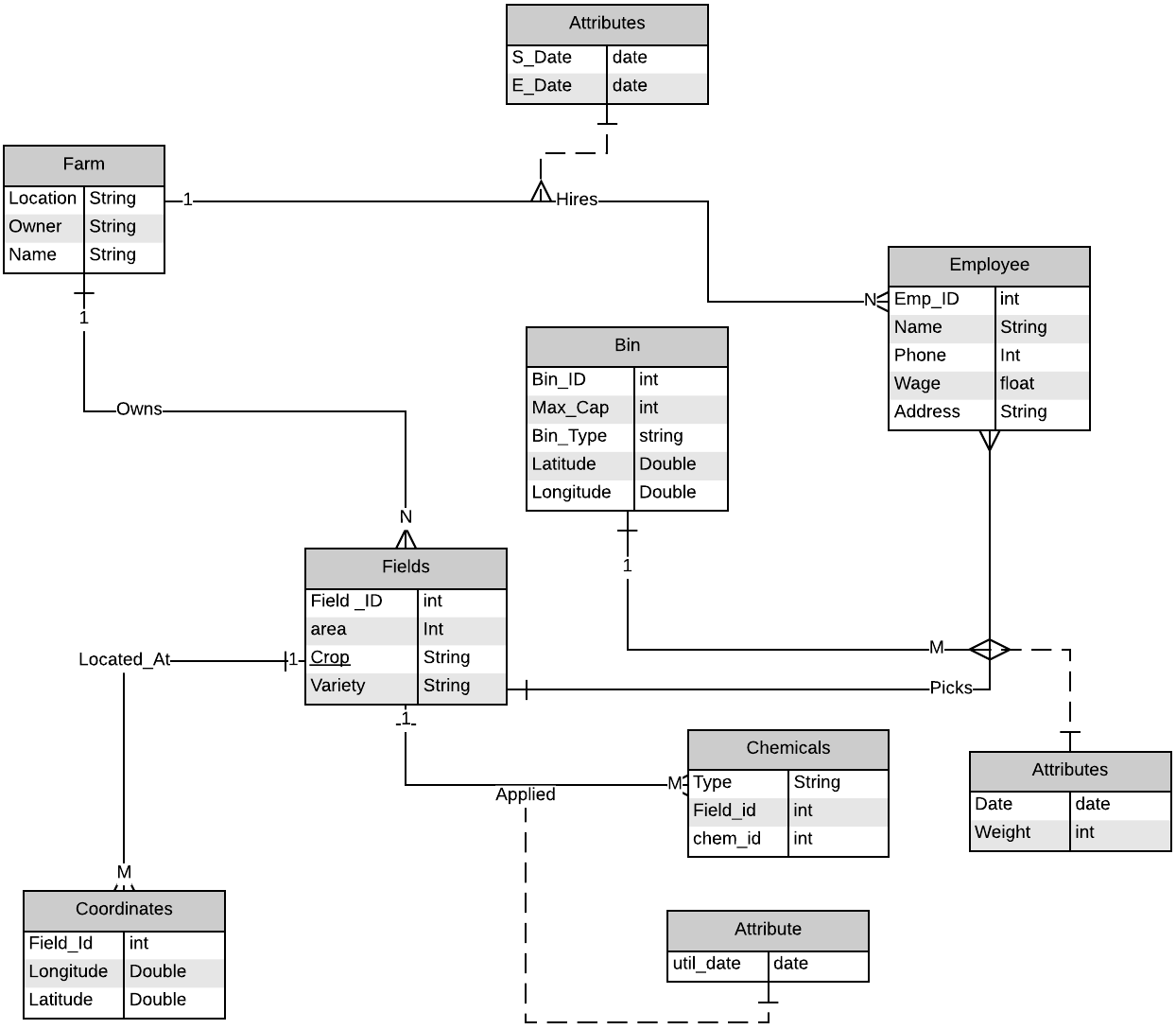
Relationship: Applied

Description: This is the relationship between fields and chemicals, meaning that fields have applied chemicals.

Entity Sets Involved	Mapping Cardinality	Descriptive Field	Participation Constraint
Fields & Chemicals	1:M	A Field can have many chemical applied	Partial

2.3 E-R Diagram





Phase 2

3. ER-Model vs. Relational Model

3.1 Description of E-R model and relational model

The entity-relationship model or better known as the ER-Model was first introduced by Peter Chen, who received his Ph.D in Computer Science and Applied

Mathematics from Harvard. This brilliant mind brought the idea of the ER Model to the Enterprise Data World which was held in 1976. Ever since then, it has became a popular high-level conceptual data model in the world of databases. Like mentioned before, and ER model is a high level data model, meaning that the developer will grab all the data that will be stored in the database and include detailed descriptions of entity types, relationships and constraints. This model can also be used to communicate with nontechnical users and to make sure that all the requirements that the user, or the person for whom the database is being developed for are met.

The ER model describes the data as entities, relationships, and attributes. Entities are an object in the real world with an independent existence. An object with a physical existence like a house, person, etc. Or even an object with a conceptual existence like an employee. Every entity will consist of attributes which are the properties that describe an entity, for example, employee name can be considered an attribute to the entity Employee because every employee has a name. There is a special type of attribute known as the key attribute which values are distinct for all the entities on the entity set. This type of attribute is unique, meaning that no two entities will have the same value for the entity. Usually each entity will have at least one key attribute, otherwise it is considered a weak entity . A relationship can be considered an association of two entities, because a relationship will link the entities it is related to. Just like entities, relationships can also contain attributes.

The relational model was first formulated by the mathematician Edgar F. Codd in 1969 .The relational model can be derived from a high-level design, like the ER model. Relational model is considered to be a logical database design, or the data model

mapping step of a database design. In this model, all data is represented as tuples, and grouped into relations. From the ER model we can get all the strong entities and make a relation for each entity E in the ER model. The key attributes in the entity will be represented as primary keys for the respective relation. In relational models, we have Foreign keys, which will provide a link between two relations.

3.2 Comparisons

The ER model is easier to develop and also it provides a lot of information about the structure of the database. This type of high-level design is a great way to start that way the developer can make sure he has covered the whole enterprise/company, before he dives into the development of the database. Whereas the relational model is more complicated to understand because it is a logical database design meaning that it is widely based on predicate calculus, tuples, and relations. The ER model might be easier to the non technical guys to understand, but once you get into querying and designing the back end of a database, the relational model will make things much more easier for you.

3.3 Converting Entity Types to Relations

To map a strong entity type from an ER-model to a relational model you will create a relation R that will include all the simple attributes of E. You will choose one of the key attributes of the entity as the primary key for your relationship R. However, if the key attribute happens to be composite, then all the attributes that form it will all be

the primary key. If the entity happens to have many keys, then you will place the keys in order to preserve uniqueness.

Another type of entity that we must worry about are the weak type entities. We convert these type of entities by first creating a relation R for the weak entity type. The relation R attributes will be the same as the weak entities' attributes and R will also contain an attribute that will serve as a foreign key that will be used to link the weak entity to the owner entity.

For each multi valued attribute we will create a relation R. The relations will include an attribute that corresponds to the multivalued attribute, and it will also include the primary key attribute of the relation for which the multivalued attribute corresponds to.

3.4 Converting Relationship Types to Relations

When it comes to converting relationship types to relations we must take inconsideration if the relationship is one-to-one, one-to-many, or many-to-many. The first approach we will look at is at the one-to-one mapping. In this particular type of relationship there are 3 approaches we can take. The first one is by using a foreign key, in this strategy we take the primary key in one relation and use it as a foreign key in another relation, that way we can have a link between the two relations. The second approach is the merged relation approach, which means to grab the relationship and the two entities that correspond to it and throw all of them in a relation. The final one is the cross-reference also known as the relationship relation approach. In this method we set up a relation R in order to cross reference the primary keys of the two entities/relations

that it belongs to. The primary key of R will be one of the foreign keys from the entities, and the other foreign key will be the unique key.

In order to map a binary one-to-many relationship we will use the entity of one side of the relationship and call it S and we will use the primary key from the other entity, which will also be a relation R, as a foreign key.

The other type of mapping we the many-to-many relationship type. To convert the relationship we will create a relation S for the relationship, and we will include as foreign keys all the primary keys in both entities that correspond to the relationship.

Another type of mapping is the N-ary relationships. This means that the relationship now maps more than two entities. We first start by creating a relation that will represent S. The foreign key attributes for S will be all of the primary keys of the relations that represent the entities that participate in the N-ary relationship.

3.5 Database Constraints

The entity constraint says that no primary key can be NULL, because having NULL values will means that there is no way for us to identify some tuples. Another important constraint is the primary key and unique key constraints. An entity must have an attribute that will distinguish it from other entities, however, sometimes an entity can contain more than one attribute that makes it unique, which we call a composite key. This means that the key will include all of the attributes.

The referential integrity constraint is another type of constraint that states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

Check constraints is another type of integrity constraint that specifies a requirement that must be met by each row in the database table. This constraint must be a predicate and it can refer to a single or multiple columns of the table.

The other type of constraint that also exists is the business rule, which cannot be expressed by schemas and must be enforced by the applications programs.

4 Relational Model

4.1 Relation Schema

Farm

Attribute Name	**Domain**	**Constraints**
Location	String	Referential
Owner	String	Referential
Name	String	Primary Key

Fields

Attribute	**Domain**	**Constraints**
Field_ID	int	Primary Key
Variety	string	Referential
longitude	int	Referential
Latitude	int	Referential
Fname	string	Referential
Crop	string	Referential

Coordinates

Attribute	Domain	Constraint
Field_Id	int	Foreign Key
Longitude	Double	Referential
Latitude	Double	Referential

Employee

Attribute	Domain	Constraints
Emp_ID	int	Primary Key
Name	string	Referential
Phone	int	Referential
Wage	Float	Referential
Address	string	Referential
F_name	string	Referential
S_date	date	Referential
E_date	date	Referential

Bin

Attribute Name	Domain	Constraints
Bin_ID	int	Primary Key
Weight	int	Referential
Bin_Type	String	Referential
Max_Cap	int	Referential

Longitude	Double	Referential
Latitude	Double	Referential

Chemicals

Attribute Name	**Domain**	**Constraints**
Chem_Id	int	Primary Key
Type	string	Referential
Field_id	int	Referential
Date	date	Referential

4.2 Sample Data

Farm

Name	**Owner**	**Location**
Valley Cherries	John Doe	Bakersfield, CA
Valley Orchards	Joe Dirt	Fresno, CA
Sunkissed Tomatoes	David Valadao	Fresno,CA
Paramount Farming	Alexander Smith	Bakersfield, CA
Cesar Chavez Farms	Jason Chi	Bakersfield, CA
Three Brothers Farming	Moises Ayala	Fresno, CA
Americas Vegetables	Brian Castagneda	Tehachapi, CA
California Farms	Omar Ramirez	Sacramento, CA
Best in the West Farms	Michael Watt	Hanford, CA
Best Vegetables	Thomas Sherman	Lemoore, CA

Fields

Fname	Field_id	longitude	latitude	Crops	Type
Jason Farming	1	0	0	Grape	Big Seed
Best Tomatoes	2	0	0	Tomato	Cherry
Bakersfield Farms	3	50	53	Grape	Seedless
Fresno Farms	4	51	50	Cherry	Black
Farms of California	5	50	50	Peach	White
Moises Farming	6	50	50	Watermelon	Seedless
Farms Of America	7	52	50	Watermelon	Seeds
Americas Ag	8	51	50	Melon	White
Agriculture for the people	9	54	50	Tomato	Regular
Paramount Farming	10	50	50	Peach	Regular

Coordinates

Field_Id	Longitude	Latitude
10000	10	20
20000	25	50
30000	35	60

40000	45	70
50000	55	80
60000	65	90
70000	75	100
80000	85	110
90000	95	120
11000	105	130

Chemicals

Chem_Id	Type	Field_id	Date
1	Pesticide	1	6/1/2014
2	Fertilizer	2	6/4/2014
3	Medicine	3	6/5/2014
4	Fertilizer	4	6/7/2014
5	Medicine	5	6/9/2014
6	Pesticide	6	6/10/2014
7	Fertilizer	6	6/11/2014
8	Medicine	6	6/15/2014
9	Pesticide	8	6/18/2014
10	Medicine	8	6/21/2014

Picks

Field_ID	Bin_Id	Emp_Id	Date	Weight
1	1	5	6/13/2014	20
2	2	2	6/17/2014	40

3	3	3	6/19/2014	32
4	4	4	6/20/2014	54
5	5	1	6/11/2014	21
6	6	8	6/23/2014	32
7	7	7	6/27/2014	54
8	8	6	6/28/2014	23
9	9	9	6/29/2014	22
10	10	10	6/30/2014	44

Bin

Bin_Id	Max_Cap	Type	Longitude	Latitude
1	200	steel	50	50
2	250	wood	52	50
3	200	wood	55	55
4	215	Steel	55	51
5	210	Wood	50	50
6	200	wood	50	50
7	205	wood	50	50
8	210	Steel	50	50
9	225	Steel	53	50
10	220	Wood	55	52

Employee

Emp_ID	F_Name	S_Date	E_date	Name	Phone	Wage	Address
1	Jason	1/7/20	6/1/2014	Juan	321-	8.00	6787 W. C++ ave.

	Farming	14			7895		Bakersfield, CA
2	Best Tomatoes	6/1/2014	NULL	Moises	123-4567	8.25	1254 W. Database ave. Bakersfield, CA
3	Bakersfield Farms	7/18/2014	NULL	Jason	321-6548	8.67	9856 W. CS ave. Bakersfield, CA
4	Fresno Farms	7/31/2014	12/12/14	Brian	789-8523	9.00	8523 W. Code ave. Bakersfield, CA
5	Farms of California	8/25/14	1/15/15	Abraham	951-2541	9.25	1235 W. Javascript ave. Bakersfield, CA
6	Moises Farming	1/20/15	3/5/15	Cesar	987-5265	9.75	5261 W. HTML5 ave. Bakersfield, CA
7	Farms Of America	1/31/15	NULL	Jose	852-9638	10.00	7532 W. Pascal ave. Bakersfield, CA
8	Americas Ag	2/14/15	3/30/15	Juan	987-4523	10.00	7852 W. Python ave. Bakersfield, CA
9	Agriculture for the people	2/28/15	NULL	Maria	852-9874	10.25	7856 W. Java ave. Bakersfield, CA
10	Paramount Farming	3/1/15	NULL	Monica	753-1452	10.75	1254 W. C# ave. Bakersfield, CA

4.3 Queries

1. **Find all farms who grew cherries with seedless variety type**

Relational Algebra:

$$\pi_{F.Name}(\sigma_{F.name = F_2.Farm_Name \wedge F_2.Type = "Seedless" \wedge F_2.Crop = "Cherries"}(Farm(F) \times Field(F_2)))$$

Tuple Relational Calculus:

$$(f \mid farms(f) \wedge \exists c(Fields(c) \wedge c.crop = "Cherry" \wedge C.Variety = "Seedless"))$$

Domain Relational Calculus:

{<N,O>|Farm<N,O> ∧ ∃crop, type(Fields(Cherry,Seedless,O,-,-,-,-,-)}

2. **Find all the fields owned by Big Run Farm who planted tomatoes type big and were picked by employee Philiip Flores.**

Relational Algebra:

$$\Pi_{Field_ID}(\sigma_{Fa.Owner = "Big\ Run\ Farm" \wedge Fa.name = F.Fname \wedge F.Type = "Big" \wedge F.Crop = "Tomatoes"}$$

$$Tfield \Leftarrow \Pi_{Field_ID}(\sigma_{Fa.Owner = "Big\ Run\ Farm" \wedge Fa.name = F.Fname \wedge F.Type = "Big" \wedge F.Crop = "Tomatoes"}$$

(Farm(F) $\times\ Farm(Fa)$)

$$Moises \Leftarrow \Pi_{Field_Id}(\sigma_{E.Emp_ID = P.Emp_ID \wedge E.name = "Moises"}(Employee(E) \times Pick(P))$$

$$\sigma_{T.Field_Id = M.Field_Id}(TField(T) \times Moises(M))$$

Tuple Relational Calculus:

(f|field(f) ∧ ∃fa∃p∃e(farm(fa) ∧ Picks(p) ∧ Employee(e) ∧ fa.owner =”Big Run Farm” ∧ f.fname=”Big Run Farm” ∧ f.crop=”Tomatoes” ∧ f.type=”Big” ∧ e.name=”Moises” ∧ p.Emp_id=e.Emp_id ∧ p.field_id = f.field_id))

Domain Relational Calculus:

{<f>|fields(-,-,-,f,-,-,-,-)∧$\exists_{owner,Name}$ (Farm(-,Big Run Farm,Jason)) ∧

$$\exists_{Tomato\ Big}(Field(Tomato,Big,-,f,-,-,-,-)_{\wedge}$$

$$\exists_{Moises}(Employee(t,Big\ Run\ Farm,-,-,Moises,-,-,-)_{\wedge}{}^{\exists}$$ (Picks(f,-,t,-,-)}

3. **Find all employee names who picked fruit into bin number 3.**

Relational Algebra:

$$\Pi_{name}(\sigma_{P.Emp_ID = E.Emp_ID \wedge P.Bin_ID = 3}(Pick(P) \times Employee(E)))$$

Tuple Relational Calculus:

$$\left(E.name \mid Employee(E) \wedge \exists\, p(Picks(p) \wedge p.Bin_Id \stackrel{!}{=} 3' \wedge P.Emp_Id = E.Emp_Id\right)$$

Domain Relational Calculus:

{<N>|Employee(I,-,-,-,N,-,-,-) ∧∃3(Pick(I,3,-,-,-) }

4. **Find all the bins that Norma Cook picked cherries into but were not the same bin as Roger Lopez.**

Relational Algebra:

$$CField \Leftarrow (\sigma_{F.Crop = "Cherries" \wedge F.Field_ID = P.Field_ID}(Field(F) \times Pick(P))$$

$$\Pi_{Bin_ID}(\sigma_{E_1.Emp_ID = C_1.Emp_ID \wedge E_1.Name = "Norma\,Cooks"}(Employee(E_1) \times CField(C_1)) -$$

$$\Pi_{Bin_ID}(\sigma_{E_2.Emp_ID = C_2.Emp_ID \wedge E_2.Name = "Robert\,Lopez"}(Employee(E_2) \times CField(C_2))$$

Tuple Relational Calculus:

(B|Bins(B) ∧ ∃e1$\exists\, p1 \exists\, e2 \exists\, p2(e1.name = "Norma\,Cook" \wedge e1.Emp_Id$=p.emp_id ∧e2.name=”Robert Lopez” ∧ p2.Emp_Id=e2.Emp_Id ∧B.bin_id=p1.Bin_id ∧p1.bin_id != p2.Bin_id))

Domain Relational Calculus

{<B>|Bin(B,-,-,-,-,-) $\wedge \exists_{Mario}$ (Employee(i,-,-,-,Norma Cook,-,-,-)$\wedge$ Picks(i,-,-,-,-) $\wedge \exists_{Cherry}$ (Cherry,-,-,i,-,-,-,-)$\wedge \exists_{John}$(Employee(j,-,-,-,Robert Lopez,-,-,-) $\wedge$ Picks(j,-,-,-,-) $\wedge \exists_{Cherry}$ (Cherry,-,-,j,-,-,-,-)$\wedge$i != j}

5. **Find all the fields who applied pesticides between 06/01/2014 through 06/15/2014.**

Relational Algebra:

$$\sigma_{F.Field_ID = C.Field_ID \wedge C.date \leq 06/15/2014 \wedge C.date \geq 06/01/2014}(Field(F) \times Chemical(C))$$

Tuple Relational Calculus:

(f|Fields(f)$\wedge \exists\, c(Chemical(c) \wedge c.type = \text{"}Pesticide\text{"} \wedge c.Date >= \text{"}06/01/2014\text{"}$

$\wedge\, f.Field_Id = c.Field_Id \wedge$ c.Date<="06/15/2014"))

Domain Relational Calculus:

$$\{<f>|Field(-,-,-,f,-,-,-,-) \wedge \exists\,(Chemicals(Pesticides,f,Date>06/01/2014\ \&\&\ Date<06/21/2014)\}$$

6. **Find the chemicals that were applied to the fields where Jason Burns worked in from 06/01/2014 to 06/21/2014.**

Relational Algebra:

$\times Pick(P)))$

$$\Pi_{type}(\sigma_{C.date \geq 06/01/2014 \wedge C.date \leq 06/21/2014}(Chemical(C) \times JField(J)))$$

Tuple Relational Calculus:

(c.type|Chemicals(c)∧∃f∃e∃p(Employee(e)∧Field(f)∧Pick(p)∧e.name=”Jason Burns” ∧ p.Date>=06/01/2014 ∧ p.Date<=06/01/2014∧ e.Emp_Id=p.Emp_Id∧ p.field_Id=f.Field_Id ∧ f.Field_Id=c.Field_Id))

Domain Relational Calculus:

{<T>|Chemicals(T,n,-) ∧∃(Employee(t,-,-,-,Jason Burns,-,-,-))∧∃ (Picks(f,-,j,Date>06/01/2014 && Date<06/21/2014,-))∧n=j }

7. **Find all employees who did not pick cherries in field 4.**

Relational Algebra:

$$\Pi_{Emp_ID}(Employee(E)) - \Pi_{Emp_ID}(\sigma_{P.Field_ID = 4 \wedge F.Field_ID = 9 \wedge F.Crop = "Cherries" \wedge P.Field_ID = F.Field_ID}(Pick(P) \times Field(F)))$$

Tuple Relational Calculus:

(e|Employee(e) ∧ ∃f(Picks(f) ∧ f.field_id!=’9’ ∧f.crop=”Cherries”∧f.emp_id=e.emp_id))

Domain Relational Calculus:

{<N>|Employee(Ei,-,-,-,N,-,-,-) ∧∃(Picks(Field_Id!=9,-,I,-,-) ∧Ei=I }

8. **Find employees who picked up buckets of fruit from field 11 on 06/09/2014 with a weight greater than 25 pounds.**

Relational Algebra:

$$\Pi\,(\sigma_{P.Emp_ID = E.Emp_ID \wedge P.Weight > 25 \wedge P.Field_ID = 11 \wedge P.Date = \text{"06/09/2014"}}(Picks(P) \times Employee(E)))$$

Tuple Relational Calculus:

$$(e \mid Employee(e) \wedge \exists\, p(Picks(p) \wedge p.weight > 25 \wedge p.Field_Id = 11 \wedge p.Date = \text{"06/09/2014"}$$

))

Domain Relational Calculus:

{<N>|Employee(j,-,-,-,N,-,-,-) ∧ ∃(Picks(11,-,i,date=06/09/2014,weight=25)∧j=i) }

9. **Find which bin was used on field 11 during 06/01/2014-06/20/2014**

Relational Algebra:

$$\Pi_{Bin_ID}(\sigma_{P.Field_ID = 11 \wedge P.Date \geq 06/01/2014 \wedge P.Date \leq 06/20/2014}(Picks(P))$$

Tuple Relational Calculus:

(b|Bin(b)∧∃p∃ f(Field(f) ∧ Picks(p) ∧ f.Field_Id=12 ∧ p.Field_Id = f.Field_Id ∧ p.Date >= “06/01/2014” ∧ p.Date<=”06/20/2014”∧ p.Bin_Id=b.Bin_Id))

Domain Relational Calculus:

{<B>|Bin(B,-,-,-,-,-) ∧ ∃(Picks(11,-,-,Date>=06/01/2014 && Date<=06/20/2014,-))∧ ∃ (Field(-,-,-,11,)) }

10. **Which fields were worked by Philip Flores and by Thomas Wilson that were not worked by Sarah Henry.**

Relational Algebra:

$$JField \Leftarrow \Pi_{Field_ID}(\sigma_{P.Emp_ID = E.Emp_ID \wedge E.name = "John" \wedge P.Date \leq 10/10/2014 \wedge P.Date \geq 9/25/2014} (Pick(P) \times Employee(E))$$

$$\Pi_{Field_Id}(\sigma_{F.Field_ID = J.Field_Id}(Field(F) \times JField(J)))$$

Tuple Relational Calculus:

(f|Fields(f) ∧ ∃e∃p(Employee(e) ∧ Picks(p) ∧ e.Name=”John” ∧ p.Emp_Id=e.Emp_Id ∧ f.Area>30 ∧ p.Date >= “9/25/2014” ∧ p.Date <= “10/10/2014” ∧ f.Crop=”Cherries”))

Domain Relational Calculus:

{<I>|Field(-,-,-,I,-,-,-) ∧∃f(Field(Cherries,-,-,j,area>30,-,-,-))∧∃ (Picks(d,-,Ei,Date>=9/25/2014 && Date<=10/10/2014))∧∃(Employee(Ed,-,-,-,John,-,-,-)) ∧j=d ∧ Ei=Ed }

Phase 3

5. Normalization of Relations

5.1 What is normalization?

Normalization of data is the process of analyzing the relation schemas based on their primary keys in order to minimize the redundancy and minimize the insertion, deletion, and update anomalies. In other words, one can think of this process of the filtering, because it will make the design have better quality. If our relation is not normalized then it might contain redundant data storage, and once our database has been fully developed, it will not organize the data in the most efficient way. Normalization plays an important role that can determine the performance of our database. The types of normalization forms that exist are first normal form (1NF), second normal form (2NF), third normal form (3NF), and Boyce-Codd Normal Form (BCNF) all these forms are described below.

1NF- Also known as first normal form, which is also known as the basic relational model. This form doesn't allow multivalued attributes, composite attributes, and their combinations. It also states that the domain of an attribute must include only atomic values and that the value in any attribute in a tuple must be a single value from the domain of that attribute. If there are any attributes that are multivalued, or any nested relations then we will form new relations for them. 1NF also disallows relations within relations or relations as attribute values within tuples. It will only permit single atomic values.

2NF- (Second Normal Form) is based on the concept of full functional dependency. Any relation schema R is said to be in 2NF if every non-prime attribute A in R is fully functionally dependent on the primary key of R. In order to normalize to this form we will set up a new relation for each partial key with its dependent attributes. However, we

must make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.

3NF-(Third Normal Form) which is the form that is based on the concept of transitive dependency. Transitive dependency means if there is a functional dependency, X onto Y in a relation schema then there exists a set of attributes Z in the relation R that is not a candidate key or a subset of R and X maps onto Z and Z maps onto Y. A relation is said to be in 3NF if it satisfies 2NF and no non prime attribute of R is transitively dependent on the primary key. In order to normalize our relation into this form, we must non key attributes.

BCNF- (Boyce-Codd Normal Form) This form was suppose to be a simpler form of 3NF but it ended up being stricter than 3NF. Which means that every relation in BCNF is set up a relation that includes the non key attributes that functionally determine other also in 3NF; but not every relation in 3NF is in BCNF. A relation schema R is said to be in BCNF form if whenever a non trivial functional dependency X onto A holds in R, then X is a superkey of R.

If the relations are not normalized then our database will run the risk of becoming inconsistent, which is also known as modification anomalies. What this means is that if we update a tuple in our table then we must make sure to update every tuple who depends on the tuple we updated otherwise our data will be unorganized or inconsistent like mentioned before.

5.3 Check Your Relations

After going back and looking at our relational schema we have analyzed and determined that all of our relations are in 1NF because our attributes in each relation are single valued and there is no repeated attributes in our relations, also a primary key has been defined, which uniquely identifies each row in the relation. We didn't have to worry about 2NF because all of our relations only contained one primary key, therefore there was no partial function dependencies, and the non key attributes depended on our key attribute. Usually relations who have two or more primary keys will usually violate the 2NF rules because there will be instances where some attributes will have partial dependencies on one primary key and the other attributes will have partial dependencies on another primary key. Therefore the developer would have to split the primary keys and their partial dependencies attributes into different relations.

After making sure all of our relations where in 1NF and in 2NF we checked if any of them had any transitivity because having transitivity in our relations would violate the 3NF rules. None of our relations had any transitivity associated with them therefore they all meet the 3NF requirements. Unfortunately none of our relation met the BCNF requirements.

5.4 What is SQL * PLUS?

SQL *Plus is an interactive and batch query tool that is installed with every Oracle Database Server or Client installation. It contains the following a command-line user interface, a Windows Graphical Interface which is also known as a GUI and it also has the iSQL *Plus web-based user interface. This utility is commonly used by users,

administrators, and programmers. SQL*Plus understands five categories of text which are:

1. SQL statements
2. PL/SQL blocks
3. SQL *Plus internal commands which are:
   1) environment control commands like SET
   2) environment monitoring commands such as SHOW
4. Comments
5. External Commands prefixed by the exclamation point.

5.5 Oracle/Schema Objects

A schema is a logical set of logical data structures. These are stored in an Oracle table space, which may exist in one or many physical files. The schema is owned by a database user and has the same name as the user. Once the data is stored in the tablespaces, data structures can be ran to manipulate data. The following data structures used are:

Tables- This is the basic unit of data storage in an Oracle database. The table is defined with a table name, and the data stored into this table is stored in rows and columns. The columns will also have a column name and a data type which contains a domain for it. A row is a collection of the column information corresponding to a single record.

An example of using sqlplus in oracle to create a new table.

```
CREATE TABLE table_name (
   Attribute_Name_1       Variable_Type_1,
```

```
   Attribute_Name_2       Variable_Type_2,
   Attribute_Name_3       Variable_Type_3,
   Table_Constraint_1,
   Table_Constraint_2
);
```

Views- Whenever you run a query on your database you will be shown a view takes the output of the query and treats it as a table. This view can be a one or more tables combined together showing any info that the query retrieves.

An example of the sqlplus command to create the database.

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Dimensions- This will define the hierarchical relation between pairs of columns or column sets.

Sequence Generator- Provides sequential series of numbers. This is really useful in multiuser environments for generating sequential numbers without the overhead of disk I/O or transaction locking. The sequence automatically generates correct values for each user.

```
CREATE SEQUENCE sequence_name (
   MINVALUE minimum_value,
   MAXVALUE maximum_value,
   START WITH initial_value,
   INCREMENT BY increment_value,
   CACHE amount_to_cache);
```

Synonyms- Is an alias for any table, view, sequence,procedure,function, or package. This requires no storage other than its definition in the data dictionary because it is only an alias.

Indexes- These are optional structures who are associated with tables and clusters. Usually indexes are created on one or more columns in a table in order to speed the SQL statement of the table. By indexing it will help you locate the information faster. However, you cannot create an index that references only one column in a table if another such index already exists.

CREATE INDEX emp_ename ON emp(ename);

Stored Procedures/Functions- These type of functions are similar to those from any high level language (C++,C,Java) because they will accept arguments and return values. Stored procedures also accept arguments, and they can return actual result sets.

CREATE OR REPLACE PROCEDURE procedure_name(var1 in var_type)

Database Linkage- This allows the data to be read-only, which means that one can't edit the data or delete it. This is usually used to see data on another DBMS without needing to login as a user in that database.

Packages- This will group logically related schema types, items and subprograms. The packages contain a specification, declared data types, variables, subprograms, and exceptions. The packages might sometimes contain a body but this is usually unnecessary.

Our database currently consists of tables, which are named: **Farm, Fields, Employee, Coordinate, Picks, Bin,** and **Chemicals.** We have also implemented stored procedures, to retrieve the data we wish to extract, for example, the employees working at a certain farm. Another schema object we have implemented is the database/linkage because we are able to pull up the table but it is only read-only, which means we can't edit it.

Insert Into Statement- This is used to insert new records into the table.

```
INSERT INTO table_name (column1,column2,column3,...)
VALUES (value1,value2,value3,...);
```

SQL Join- ins are used to combine rows from two or more tables.

Max()- Will return the maximum value from the column

```
SELECT MAX(column_name) FROM table_name;
```

Having clause- This is similar to the where command but is compatible with aggregated functions.

```
HAVING aggregate_function(column_name) operator value;
```

Group by- statement is used in conjunction with the aggregate functions to group the result-set by one or more columns

```
GROUP BY column_name;
```

Select- This statement is used to select data from a database

```
SELECT column_name,column_name
FROM table_name;
```

Where- This statement is used to filter records.

```
WHERE column_name operator value;
```

Drop-The DROP INDEX statement is used to delete an index in a table.

```
DROP INDEX index_name
```

Purge Recyclebin- This cleans out the recycle bin which will remove any junk tables from the database.

```
Purge Recyclebin;
```

5.6 Relation Schema

Shown below are the responses from SELECT and DESC queries done on our table.

select * from jcma_farm;

```
ORCHARD SQL> select * from jcma_farm;

FARM_NAME                      LOCATION                       OWNER
------------------------------ ------------------------------ ------------------------------
A Plus Carson Ranch            915 Roth Hill                  David Carson
Bailey Farms                   7178 Dixon Lane                Irvin Bailey
Ark Park Acres                 84310 Green Lane               Tracy Tritle
Big Run Farm                   3545 Westport Hill             Linda Gannon
Grab Bag Farm                  89858 Pierstorff Park          Ellen Winger
Lazy Dog Ranch                 3 Buell Trail                  Cheryl Oliver
Molasses Meadow                975 South Road                 Glen Smith
Moonlight Meadows              2 Sauthoff Place               Bonnie Stone
Parkway Farms                  18780 Fulton Drive             Barron Buchanan
Roadrunner Donkey Farm         4 Dottie Road                  Elaine Warlick

10 rows selected.
```

desc jcma_farm;

```
ORCHARD SQL> desc jcma_farm
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------------
 FARM_NAME                                                 NOT NULL VARCHAR2(32)
 LOCATION                                                           VARCHAR2(32)
 OWNER                                                              VARCHAR2(32)
```

select * from jcma_field

```
ORCHARD SQL> select * from jcma_fields;

  FIELD_ID FARM_NAME                       CROP                             VARIETY
---------- ------------------------------ -------------------------------- ----------
         1 A Plus Carson Ranch            cherry                           seedless
         2 A Plus Carson Ranch            cherry                           seed
         3 Bailey Farms                   watermelon                       seedless
         4 Bailey Farms                   watermelon                       seed
         5 Ark Park Acres                 peach                            red
         6 Ark Park Acres                 peach                            white
         7 Big Run Farm                   tomato                           big
         8 Big Run Farm                   tomato                           small
         9 Grab Bag Farm                  cherry                           seedless
        10 Grab Bag Farm                  cherry                           seed
        11 Lazy Dog Ranch                 melon                            white
        12 Lazy Dog Ranch                 melon                            orange
        13 Molasses Meadow                onion                            red
        14 Molasses Meadow                onion                            white
        15 Moonlight Meadows              watermelon                       seed
        16 Moonlight Meadows              watermelon                       seedless
        17 Parkway Farms                  melon                            white
        18 Parkway Farms                  melon                            orange
        19 Roadrunner Donkey Farm         almond                           big
        20 Roadrunner Donkey Farm         almond                           small

20 rows selected.
```

desc jcma_field

```
ORCHARD SQL> desc jcma_fields
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------------------------
 FIELD_ID                                                  NOT NULL NUMBER(38)
 FARM_NAME                                                          VARCHAR2(32)
 CROP                                                               VARCHAR2(32)
 VARIETY                                                            VARCHAR2(32)
```

select * from jcma_employee

```
ORCHARD SQL> select * from jcma_employee;

    EMP_ID NAME                           PHONE           ADDRESS
---------- ------------------------------ --------------- ------------------------
FARM_NAME                      S_DATE    E_DATE
------------------------------ --------- ---------
         1 Norma Cook                     42059862784     57 Hermina Alley
A Plus Carson Ranch            13-AUG-09 16-MAR-14

         2 Irene Simpson                  60773010068     541 Hansons Point
A Plus Carson Ranch            20-NOV-09 29-OCT-14

         3 Jason Burns                    21106305527     6364 6th Court
Bailey Farms                   01-JUN-09 23-FEB-14

         4 Nicole Watkins                 10103089343     2446 Burning Wood Point
Bailey Farms                   27-APR-09

         5 Sarah Henry                    69247205729     65 Hoard Circle
Ark Park Acres                 19-FEB-09 08-OCT-13

         6 Janet Wagner                   30349008579     6 Eagan Circle
Ark Park Acres                 17-AUG-09 20-MAR-13

         7 Howard Ruiz                    45701313547     56045 Brentwood Parkway
Big Run Farm                   01-JAN-10 13-NOV-13

         8 Phillip Collins                89142500806     46783 Jenna Pass
Big Run Farm                   18-AUG-09

         9 Steven Sanders                 82019357631     99 Larry Alley
Grab Bag Farm                  04-MAR-10 25-DEC-12

        10 Roger Lopez                    74876002202     271 Mayer Lane
Grab Bag Farm                  30-JUL-10 23-JUL-14

        11 Arthur Meyer                   45203870109     253 Union Trail
Lazy Dog Ranch                 02-MAR-09 08-AUG-14

        12 Philip Flores                  76184979677     9034 Walton Way
Lazy Dog Ranch                 14-OCT-09
```

continued…

```
    EMP_ID NAME                           PHONE           ADDRESS
---------- ------------------------------ --------------- ----------------------------
FARM_NAME                       S_DATE    E_DATE
------------------------------- --------- ---------

        13 Philip Harper                  40557121311     109 Nova Junction
Molasses Meadow                 27-APR-09 16-APR-13

        14 Thomas Wilson                  69110854334     111 Golf View Plaza
Molasses Meadow                 17-JUN-09 20-AUG-13

        15 Doris Robertson                92614981385     63364 Iowa Circle
Moonlight Meadows               07-AUG-10

        16 Randy Arnold                   30063357773     7 Emmet Court
Moonlight Meadows               08-AUG-09 03-JUN-14

        17 Carl Gardner                   49404320927     8 Susan Trail
Parkway Farms                   04-JUL-09

        18 Ashley Hart                    59053436589     53 Fieldstone Court
Parkway Farms                   13-JUN-10 19-AUG-14

        19 Phillip Phillips               39358163503     7558 Center Hill
Roadrunner Donkey Farm          04-SEP-10 03-JAN-13

        20 Phyllis Peters                 04765593678     891 Mallory Alley
Roadrunner Donkey Farm          24-OCT-10 11-APR-14


20 rows selected.
```

desc jcma_employee

```
ORCHARD SQL> desc jcma_employee
 Name                                                      Null?    Type
 --------------------------------------------------------- -------- ------------
 EMP_ID                                                    NOT NULL NUMBER(38)
 NAME                                                               VARCHAR2(32)
 PHONE                                                              VARCHAR2(16)
 ADDRESS                                                            VARCHAR2(32)
 FARM_NAME                                                          VARCHAR2(32)
 S_DATE                                                             DATE
 E_DATE                                                             DATE
```

select * from jcma_coordinates

```
ORCHARD SQL> select * from jcma_coordinates;

  FIELD_ID  LONGITUDE   LATITUDE
---------- ---------- ----------
         7  -121.7053    37.1894
        12     7.7923    63.1103
         2  114.95897   23.77894
        14     1.4437    43.6043
        19  -75.70028  -14.03583
        12  123.73033    9.58745
        14  123.50851   12.22854
        16    71.8667   30.40677
         6    -98.175    22.2485
         4   112.0324    -8.2667
        12  106.50606   10.15021
         9  121.39028    5.95944
        12   29.00974  -24.19436
        19   44.54587  -19.69787
         4  120.34029   40.33468
         9      25.65      43.15
         4     83.521    53.4203
        16   22.52139   43.88778
        14  -72.13611  -14.30778
        20   34.37167   53.25209
         2   95.97965    21.8787
         8  -71.24087    8.54665
        10   -8.53778    7.57944
        20   19.91667   40.80028
         9  110.32889   -7.88806
         5   36.84085   53.43852
         9   96.54339   48.98562
         5  -85.46218   12.83734
         5  -60.47389  -34.64167
        10    42.3323   56.52801
        11   35.23784   32.10841
        20   19.26737   50.75185
         6  120.43482   29.32999
         4   -97.1164    20.0747
        14   30.61534    59.9355
        16  111.40418   29.59085
         1    111.948   24.06556
```

continued……

FIELD_ID	LONGITUDE	LATITUDE
11	57.6996	27.9519
3	122.08205	13.31676
11	125.81667	7.83333
15	-4.87228	33.85194
3	5.1092	45.6305
1	121.56766	50.6351
15	-108.43471	54.13348
3	118.47292	29.46301
11	112.6975	28.62833
15	11.9294	57.6467
1	117.707	-8.6715
18	111.70194	25.53417
6	118.98023	31.60732
1	16.37992	6.31933
16	17.1015	58.6706
10	106.4	47.7
18	22.39976	40.58873
7	-73.96694	-15.80611
17	-70.58889	-14.88639
8	113.4586	-8.3081
19	-42.22222	-14.50333
8	110.12515	29.18713
17	17.61932	50.642
6	-43.48722	-19.94583
19	82.6712	53.5044
3	19.65512	44.77106
2	55.97806	55.035
17	112.75806	-7.38194
13	105.9254	-6.1987
13	-70.68333	-16.75
17	21.02368	49.39598
5	22.46601	50.29976
7	32.40557	-18.96167
15	130.97478	-1.41179
10	125.39917	6.59583
8		
13	-1.5497	47.1814

continued….

FIELD_ID	LONGITUDE	LATITUDE
20	-100.9223	28.4932
18	121.2146	31.03595
7	99.94467	13.11189
13	-22.56111	63.99813
18	14.04037	50.18844
2	27.78022	-25.63473
7	-121.7053	37.1894
12	7.7923	63.1103
2	114.95897	23.77894
14	1.4437	43.6043
19	-75.70028	-14.03583
12	123.73033	9.58745
14	123.50851	12.22854
16	71.8667	30.40677
6	-98.175	22.2485
4	112.0324	-8.2667
12	106.50606	10.15021
9	121.39028	5.95944
12	29.00974	-24.19436
19	44.54587	-19.69787
4	120.34029	40.33468
9	25.65	43.15
4	83.521	53.4203
16	22.52139	43.88778
14	-72.13611	-14.30778
20	34.37167	53.25209
2	95.97965	21.8787
8	-71.24087	8.54665
10	-8.53778	7.57944
20	19.91667	40.80028
9	110.32889	-7.88806
5	36.84085	53.43852
9	96.54339	48.98562
5	-85.46218	12.83734
5	-60.47389	-34.64167
10	42.3323	56.52801
11	35.23784	32.10841

continued….

FIELD_ID	LONGITUDE	LATITUDE
20	19.26737	50.75185
6	120.43482	29.32999
4	-97.1164	20.0747
14	30.61534	59.9355
16	111.40418	29.59085
1	111.948	24.06556
11	57.6996	27.9519
3	122.08205	13.31676
11	125.81667	7.83333
15	-4.87228	33.85194
3	5.1092	45.6305
1	121.56766	50.6351
15	-108.43471	54.13348
3	118.47292	29.46301
11	112.6975	28.62833
15	11.9294	57.6467
1	117.707	-8.6715
18	111.70194	25.53417
6	118.98023	31.60732
1	16.37992	6.31933
16	17.1015	58.6706
10	106.4	47.7
18	22.39976	40.58873
7	-73.96694	-15.80611
17	-70.58889	-14.88639
8	113.4586	-8.3081
19	-42.22222	-14.50333
8	110.12515	29.18713
17	17.61932	50.642
6	-43.48722	-19.94583
19	82.6712	53.5044
3	19.65512	44.77106
2	55.97806	55.035
17	112.75806	-7.38194
13	105.9254	-6.1987
13	-70.68333	-16.75
17	21.02368	49.39598

continued..

```
  FIELD_ID  LONGITUDE   LATITUDE
---------- ---------- ----------
         5   22.46601   50.29976
         7   32.40557  -18.96167
        15  130.97478   -1.41179
        10  125.39917    6.59583
         8
        13    -1.5497    47.1814
        20  -100.9223    28.4932
        18   121.2146   31.03595
         7   99.94467   13.11189
        13  -22.56111   63.99813
        18   14.04037   50.18844
         2   27.78022  -25.63473

160 rows selected.
```

select * from jcma_chemicals

```
ORCHARD SQL> select * from jcma_chemicals;

   CHEM_ID CHEMICAL_TYPE                    FIELD_ID UTIL_DATE
---------- -------------------------------- ---------- ---------
         1 Fertilizer                                1 01-JUN-14
         2 Pesticides                                2 03-JUN-14
         3 Fertilizer                                5 10-JUN-14
         4 Pesticides                                5 15-JUN-14
         5 Fertilizer                               10 15-JUN-14
         6 Pesticides                               11 15-JUN-14
         7 Fertilizer                               12 19-JUN-14
         8 Fertilizer                               13 21-JUN-14
         9 Pesticides                               15 23-JUN-14
        10 Fertilizer                               20 25-JUN-14
        11 Pesticides                               20 29-JUN-14

11 rows selected.
```

desc jcma_chemicals

```
ORCHARD SQL> desc jcma_chemicals
 Name                                                      Null?    Type
 --------------------------------------------------------- -------- ------------
 CHEM_ID                                                   NOT NULL NUMBER
 CHEMICAL_TYPE                                                      VARCHAR2(32)
 FIELD_ID                                                           NUMBER(38)
 UTIL_DATE                                                          DATE
```

select * from jcma_picks

```
ORCHARD SQL> select * from jcma_picks;

  FIELD_ID     BIN_ID     EMP_ID P_DATE       WEIGHT
---------- ---------- ---------- --------- ----------
         9          6          1 16-NOV-13       4.03
         6         10         17 13-MAR-13      29.69
         5          1         12 16-SEP-14      11.31
        11          4         12 09-MAY-14       1.61
        11         10          4 23-JAN-13       6.06
        17          5         10 26-JUN-13      13.39
        19          2          1 06-JUN-13      27.17
         8          3          2 24-APR-14       7.63
        19          8         11 27-SEP-14      12.05
        18         10         20 27-OCT-14      29.37
        16          6         19 11-FEB-13       4.17
        14          2         13 08-APR-13        7.3
        11          2         17 16-SEP-13      10.66
         2          4         14 12-AUG-14      24.26
         1          9          1 12-APR-13      26.52
         1          5          2 05-DEC-13       8.31
         8          6         13 06-DEC-12         19
         1          4         16 15-MAR-14      18.19
        13          4          5 16-MAR-14       6.04
         7          3          8 30-JUL-14      14.75
        12          7          2 06-JUN-13       17.5
        14         10          4 22-MAY-14       13.8
        11          2         17 11-JUL-13      13.41
        18          6          6 07-MAY-14      16.07
         8         10         11 17-JUL-14      11.84
        12          3         13 01-NOV-13       4.49
        10          7         12 13-MAY-13      11.09
         4          9         19 11-OCT-14      24.57
         9          6         10 01-MAR-13       4.03
        11          8         16 21-AUG-13      26.01
         9          1          1 19-MAY-13      11.06
         9          1         12 17-APR-14       9.73
        18          7          9 17-MAR-13       3.14
        13         10          4 10-AUG-13       1.63
        14         10          3 30-JUL-14       3.09
         7          3          8 23-MAR-13       6.79
         7          1          3 29-MAY-14       9.56
```

continued…

FIELD_ID	BIN_ID	EMP_ID	P_DATE	WEIGHT
5	3	6	17-MAR-13	25.93
17	8	2	06-JUL-13	12.4
2	3	4	06-SEP-13	22.52
1	1	12	15-AUG-14	17.11
2	2	13	22-MAY-14	5.46
14	1	8	29-DEC-12	25.38
13	4	13	08-APR-14	9.13
12	4	17	28-MAY-14	24.38
12	7	13	28-SEP-13	5.65
13	4	5	27-DEC-13	15.73
15	5	15	09-APR-14	23.67
11	5	17	19-AUG-14	15.18
3	4	14	12-NOV-12	7.17
4	7	10	25-FEB-13	15.94
8	4	3	13-JUN-13	22.3
3	3	12	16-APR-13	2.46
7	4	14	22-JAN-13	27.98
11	6	10	11-NOV-13	3.96
1	7	18	20-DEC-12	29.79
11	6	10	27-JUN-14	7.15
9	9	19	05-JAN-14	29.87
4	8	8	24-SEP-13	28.47
13	10	6	22-FEB-13	13.11
2	8	8	26-NOV-12	20.98
6	5	4	18-FEB-14	15.99
6	5	9	26-APR-13	19.71
12	6	20	01-JAN-13	13.5
16	2	13	09-OCT-13	21.69
3	7	12	04-OCT-14	4.25
6	8	18	24-AUG-14	6.79
19	7	1	23-APR-14	23.62
17	3	9	15-OCT-14	9.97
7	8	12	16-AUG-14	26
19	9	3	19-MAR-13	19.68
7	3	12	11-SEP-13	25.57
4	7	18	13-JUL-13	6.5
4	8	1	02-DEC-13	5.13

continued..

```
  FIELD_ID     BIN_ID     EMP_ID P_DATE        WEIGHT
---------- ---------- ---------- --------- ----------
         4          6          2 08-APR-13      19.18
         8          8          2 09-OCT-13      25.31
         4          9         12 07-NOV-13      17.14
        16          1         11 09-MAY-14      19.38
         9          3         11 11-APR-13      28.66
        13          9          1 06-MAR-14      17.06

80 rows selected.
```

desc jcma_picks;

```
ORCHARD SQL> desc jcma_picks;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------
 FIELD_ID                                                           NUMBER(38)
 BIN_ID                                                             NUMBER(38)
 EMP_ID                                                             NUMBER(38)
 P_DATE                                                             DATE
 WEIGHT                                                             NUMBER
```

select * from jcma_bin;

```
ORCHARD SQL> select * from jcma_bin;

    BIN_ID     WEIGHT BIN_TYPE                         LATITUDE  LONGITUDE    MAX_CAP
---------- ---------- ------------------------------ ---------- ---------- ----------
         1        287 cherry                               20.2    -120.32        500
         2        187 onion                                20.2    -120.32        400
         3        254 watermelon                           20.2    -120.32        500
         4        248 almond                               20.2    -120.32        300
         5        261 cherry                               20.2    -120.32        400
         6        300 melon                                20.2    -120.32        300
         7        500 cherry                               20.2    -120.32        500
         8        315 peach                                20.2    -120.32        400
         9        450 cherry                               20.2    -120.32        500
        10        443 cherry                               20.2    -120.32        500

10 rows selected.
```

desc jcma_bin

```
ORCHARD SQL> desc jcma_bin
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- --------------
 BIN_ID                                                    NOT NULL NUMBER(38)
 WEIGHT                                                             NUMBER(8)
 BIN_TYPE                                                           VARCHAR2(32)
 LATITUDE                                                           NUMBER(38,13)
 LONGITUDE                                                          NUMBER(38,13)
 MAX_CAP                                                            NUMBER(8)
```

5.7 SQL Queries

The following queries were developed by us.

1) Find all the farms that grew ‘seedless’ variety of cherries.

```
select distinct f1.farm_name from jcma_farm f1
join jcma_fields f2 on f1.farm_name = f2.farm_name
where
f2.crop = 'cherry' and
f2.variety = 'seedless'
/
```

Result:

```
ORCHARD SQL> @qry01

FARM_NAME
----------------------------
Grab Bag Farm
A Plus Carson Ranch
```

2) Find all the fields that is owned by ‘Big Run Farm’ that grew ‘big’ tomatoes variety and had employee ‘Philip Flores’ working on that field.

```
select distinct f.field_id from jcma_fields f
join jcma_picks p on f.field_id = p.field_id
join jcma_employee e on e.emp_id = p.emp_id
where
e.name = 'Philip Flores' and
f.crop = 'tomato' and
f.variety = 'big' and
f.farm_name = 'Big Run Farm'
/
```

Result:

```
ORCHARD SQL> @qry02

  FIELD_ID
----------
         7
```

3) Find the name of all the employees that have deposited fruit into bin number 3.

```
select distinct e.name from jcma_employee e
join jcma_picks p on p.emp_id = e.emp_id
where
p.bin_id='3'

/
```

Result:

```
ORCHARD SQL> @qry03

NAME
--------------------------------
Arthur Meyer
Steven Sanders
Irene Simpson
Nicole Watkins
Philip Flores
Jason Burns
Janet Wagner
Phillip Collins
Philip Harper

9 rows selected.
```

4) Find all the fields that ‘Norma Cook’ has work on but exclude the fields that ‘Roger Lopez’ has work on.

```
select distinct p.bin_id from jcma_picks p
join jcma_employee e on e.emp_id = p.emp_id
join jcma_fields f on f.field_id = p.field_id
where (
    f.crop = 'cherry' and
    e.name = 'Norma Cook'
)
MINUS
select distinct p2.bin_id from jcma_picks p2
join jcma_employee e2 on e2.emp_id = p2.emp_id
join jcma_fields f2 on f2.field_id = p2.field_id
where (
    f2.crop = 'cherry' and
    e2.name = 'Roger Lopez'
);
```

Result:

```
ORCHARD SQL> @qry04

    BIN_ID
----------
         1
         9
```

5) Find all the field that have had applied Pesticides between the dates of 06/01/2014 and 06/15/2014.

```
select f.field_id from jcma_fields f
join jcma_chemicals c on f.field_id = c.field_id
where
c.util_date >= TO_DATE('6/01/2014', 'mm/dd/yyyy') and
c.util_date <= TO_DATE('6/15/2014', 'mm/dd/yyyy') and
c.chemical_type='Pesticides'
/
```

Result:

```
ORCHARD SQL> @qry05

  FIELD_ID
----------
         2
         5
        11
```

6) Find which chemicals was applied on the field that Jason Burns was working on during the period of 06/01/2014 and 06/21/2014.

```
select c.chemical_type from jcma_chemicals c
join jcma_picks p on p.field_id=c.field_id
join jcma_employee e on e.emp_id=p.emp_id
where
e.name='Jason Burns' and
p.p_date>=TO_DATE('2014-06-01 00:00:00','yyyy-mm-dd hh24:mi:ss') and
p.p_date<=TO_DATE('2014-06-21 23:59:59','yyyy-mm-dd hh24:mi:ss') and
c.util_date >= TO_DATE('6/01/2014','mm/dd/yyyy') and
c.util_date <= TO_DATE('6/21/2014','mm/dd/yyyy')
/
```

Result:

```
ORCHARD SQL> @qry06

CHEMICAL_TYPE
--------------------------
Pesticides
Pesticides
Pesticides
Fertilizer
Pesticides
```

7) Find all the employees who did not pick cherries in field 9.

```
select distinct e.emp_id from jcma_employee e
MINUS
select distinct p.emp_id from jcma_picks p
join jcma_fields f on f.field_id = p.field_id
where (
    f.crop = 'cherry' and
    f.field_id = '9'
)
/
```

Result:

```
ORCHARD SQL> @qry07

    EMP_ID
----------
         2
         3
         4
         5
         6
         7
         8
         9
        13
        14
        15
        16
        17
        18
        20

15 rows selected.
```

8) Find all the employees who picked at least 25 pounds on field 11 on 06/09/2014

```
select e.name from jcma_employee e
join jcma_picks p on p.emp_id = e.emp_id
where
    p.field_id = '11' and
    p.weight >= 25 and
    p.p_date >= TO_DATE('2014-06-09 00:00:00','yyyy-mm-dd hh24:mi:ss') and
    p.p_date <= TO_DATE('2014-06-09 23:59:59','yyyy-mm-dd hh24:mi:ss')

/
```

Result:

```
ORCHARD SQL> @qry08

NAME
------------------------------
Norma Cook
Jason Burns
Janet Wagner
```

9) Find the bin used on field 11 during 06/01/2014 and 06/20/2014.

```
select b.bin_id from jcma_bin b
join jcma_picks p on p.bin_id=b.bin_id
where
p.field_id='11' and
p.p_date >= TO_DATE('2014-06-01 00:00:00','yyyy-mm-dd hh24:mi:ss') and
p.p_date <= TO_DATE('2014-06-20 23:59:59','yyyy-mm-dd hh24:mi:ss')
/
```

Result:

```
ORCHARD SQL> @qry09

    BIN_ID
----------
        10
         5
         3
         9
```

10) Which fields were worked by Philip Flores and by Thomas Wilson that were not worked by Sarah Henry.

```
(
select f.field_id from jcma_fields f
join jcma_picks p on p.field_id=f.field_id
join jcma_employee e on e.emp_id=p.emp_id
where(
e.name='Philip Flores'
)
intersect
select f2.field_id from jcma_fields f2
join jcma_picks p2 on p2.field_id=f2.field_id
join jcma_employee e2 on e2.emp_id=p2.emp_id
where(
e2.name='Thomas Wilson'
)
)
minus
select f3.field_id from jcma_fields f3
join jcma_picks p3 on p3.field_id=f3.field_id
join jcma_employee e3 on e3.emp_id=p3.emp_id
where(
e3.name='Sarah Ham'
)
/
```

Result:

```
ORCHARD SQL> @qry10

  FIELD_ID
----------
         3
         7
```

11) What is the maximum weight that Philip Flores has picked throughout his career.

```
select MAX(p.weight) from jcma_picks p
join jcma_employee e on p.emp_id = e.emp_id
where
e.name='Philip Flores'
/
```

Result:

```
ORCHARD SQL> @qry12

MAX(P.WEIGHT)
-------------
           26
```

12) Find the total fruit collected for the field which their ID is greater than 15

```
select field_id, SUM(weight) from jcma_picks
group by field_id
having field_id > 15
order by field_id
/
```

Result:

```
ORCHARD SQL> @qry11

  FIELD_ID SUM(WEIGHT)
---------- -----------
        16       45.24
        17       35.76
        18       48.58
        19       62.84
```

13) List all the bins that have a pick associated with it.

```
select * from jcma_bin b
where exists (
    select *
    from jcma_picks p
    where p.bin_id = b.bin_id
)
/
```

Result:

```
ORCHARD SQL> @qry13

    BIN_ID     WEIGHT BIN_TYPE                         LATITUDE  LONGITUDE    M
AX_CAP
---------- ---------- ------------------------------ ---------- ---------- ----
------
         6        300 melon                                20.2    -120.32
   300
        10        443 cherry                               20.2    -120.32
   500
         1        287 cherry                               20.2    -120.32
   500
         4        248 almond                               20.2    -120.32
   300
         5        261 cherry                               20.2    -120.32
   400
         2        187 onion                                20.2    -120.32
   400
         3        254 watermelon                           20.2    -120.32
   500
         8        315 peach                                20.2    -120.32
   400
         9        450 cherry                               20.2    -120.32
   500
         7        500 cherry                               20.2    -120.32
   500

10 rows selected.
```

Phase 4

6.1 Common Features of PL/SQL and T-SQL

Procedural Language/SQL(PL/SQL) and Transact SQL(T-SQL) both have many similarities because they are based on the original SQL. However, since they are being developed by competing companies, Oracle and Microsoft respectively, these

languages can have differing implementation of similar features. For example, both languages have similar support query commands like the select command or join command. Another commonality between the two languages is that they store the trigger and procedures to be stored on the server. Since both languages are based on SQL, they include clauses, expressions, predicates, queries, and statements from SQL.

The differences between the two languages can vary a lot from small syntax differences to large differences like supporting differing functions. One large difference between PL/SQL and Transact SQL is that Oracle has Packages as subroutines. There are no packages in Transact SQL. Packages are an object that encapsulates statements, object, subprograms, and variables. They are similar in concept to classes in C# or C++ because it allows functions within the package to use shared variables. To implement functionality similar to packages in Transact SQL, the user would have duplicate data within the implementation due to not having shared variables. Packages are also stored in the database along with the other subroutines.

6.2 Purpose of Stored Subprograms

In PL/SQL, subprograms includes functions, procedures and packages. They have the ability to take arguments in and can be comprised of a set of complicated commands. The purpose of subprograms is to allow the user to encapsulate complicated procedures to improve reusability of code. It can also provide control of how certain actions are performed. For example, in the current database there are insert procedures that contain sequences that increment the ID for each new row. Subprograms are more efficient when compared to running commands by statements.

6.3 Benefits of calling stored subprogram over sending a dynamic SQL

It is more efficient to use stored subprograms due to to the fact that subprograms are precompiled on the server and can take a single transaction call to complete a procedure. In comparison, if a same instruction set is sent over dynamic SQL then each statement is considered a transaction. So, it takes more transactions to complete and that will take more time. Subprograms can make common procedures or automated calls more efficient. In addition to this, subprograms also provide reusability and maintainability, because once validated, a subprogram can be used with confidence in any number of applications.

6.4 Oracle PL/SQl

PL/SQL consists of program structure, control statements, and cursors. A program structure is a block that consists of three parts which are:

1. **Declaration**-declare the variables, constraints, cursors, and user-defined expressions.
2. **Executable**-which consists of SQL/SQLPLUS statements.
3. **Exception Handling**- A predefined or user-defined warning or error that is handled by the PL/SQL program.

The control statement consists of conditional, iterative, and sequential controls. Conditional controls are “if/else if” statements, whereas iterative controls are loops like the “while” and the “for” loop.

Cursors are used by database programmers to retrieve specified rows based on the database system queries. A cursor will enable the manipulation of whole result sets all at once.

Stored Procedure:

A stored procedure will perform one or more specified tasks. It contains a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists the execution section and the exception section. A stored procedure may or may not return a value.

Syntax For Stored Procedure In PL/SQL:

```
CREATE [OR REPLACE] PROCEDURE proc_name [list of parameters]
IS
  Declaration section
BEGIN
  Execution section
EXCEPTION
 Exception section
END;
```

Syntax For Stored Procedure in Trans-SQL:

```
CREATE PROC [ EDURE ] procedure_name [ ;number ]
        [ { @parameter data_type }
        [ VARYING ] [ =default ] [ OUTPUT ]
        ] [ ,... ]
[ WITH
```

{

RECOMPILE

| ENCRYPTION

| RECOMPILE, ENCRYPTION

}

]

[FOR REPLICATION]

AS

sql_statement

Stored Procedures in PL/SQL vs Trans-SQL Syntax:

In the Trans-SQL syntax, there is the Recompile, Encryption, and the Recompile Encryption and this is not included in the syntax for the PL/SQL. However in the PL/SQL we have the the Exception block, which can output an error to the screen whenever an exception happens.

Stored Function:

A stored function is similar to a stored procedure, but the main difference between these two, is that a function must always return a value and a procedure may or may not return a value.

Syntax For Stored Function In PL/SQL:

CREATE [OR REPLACE] FUNCTION function_name [parameters]

RETURN return_datatype;

IS

Declaration_section

BEGIN

Execution_section

Return return_variable;

EXCEPTION

exception section

Return return_variable;

END;

Stored Function in Trans-SQL:

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
   [ = default ] [ READONLY ] }
   [ ,...n ]
 ]
)
RETURNS return_data_type
   [ WITH <function_option> [ ,...n ] ]
   [ AS ]
   BEGIN
      function_body
      RETURN scalar_expression
   END
[ ; ]
```

Stored Function in PL/SQL vs. Trans-SQL syntax:

In the Trans-SQL syntax, one must provide the name of the schema before the function name. Also the Trans-SQL does not provide us with the “OR REPLACE” command. PL/SQL also has a place for the exception that will print out an error whenever an exception is fired.

Package:

A package is a schema object that groups logically related PL/SQL types, items, and subprograms. Packages usually have two parts a specification and a body. The specification declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The body fully defines cursors and subprograms, and so implements the specifications.

Syntax For Package In PI/SQL:

Defining Package Specification Syntax

```
CREATE [OR REPLACE] PACKAGE package_name
   [AUTHID {CURRENT_USER | DEFINER}]
   {IS | AS}
   [PRAGMA SERIALLY_REUSABLE;]
   [collection_type_definition ...]
   [record_type_definition ...]
   [subtype_definition ...]
   [collection_declaration ...]
   [constant_declaration ...]
   [exception_declaration ...]
   [object_declaration ...]
   [record_declaration ...]
   [variable_declaration ...]
   [cursor_spec ...]
   [function_spec ...]
   [procedure_spec ...]
```

```
  [call_spec ...]
  [PRAGMA RESTRICT_REFERENCES(assertions) ...]
END [package_name];
```

Creating Package Body Syntax In PL/SQL

```
[CREATE [OR REPLACE] PACKAGE BODY package_name {IS | AS}
  [PRAGMA SERIALLY_REUSABLE;]
  [collection_type_definition ...]
  [record_type_definition ...]
  [subtype_definition ...]
  [collection_declaration ...]
  [constant_declaration ...]
  [exception_declaration ...]
  [object_declaration ...]
  [record_declaration ...]
  [variable_declaration ...]
  [cursor_body ...]
  [function_spec ...]
  [procedure_spec ...]
  [call_spec ...]
[BEGIN]
  sequence_of_statements]
END [package_name];]
```

Reminder: We can’t create packages in Trans-SQl.

Triggers:

A trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is fired automatically when an associated DML statement is executed.

Syntax For Triggers In PL/SQL:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
BEGIN
  --- sql statements
END;
```

Syntax for Triggers in Trans-SQL:

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }
```

<dml_trigger_option> ::=
[ENCRYPTION]
[EXECUTE AS Clause]

<method_specifier> ::=
assembly_name.class_name.method_name

Syntax for Triggers in PL/SQL vs. Trans-SQL

Triggers in Trans-SQL are different than the ones in PL/SQL because in Trans-SQL the triggers can't be used with the "BEFORE" command. The Trans-SQL only contains an "AFTER" and an "INSTEAD OF" command. The Syntax for the Trans-SQL also contains an "Encryption" and an "Execute As Clause" command.

6.5 Oracle PL/SQL Subprogram

Stored Procedures:

Insert farm

This procedure takes 3 parameters that are the same data type as the fields of the farm table and will be inserted as a new row.

```
CREATE OR REPLACE PROCEDURE insert_farm (
    ins_farm_name in jcma_farm.farm_name%TYPE,
    ins_location  in jcma_farm.location%TYPE,
    ins_owner     in jcma_farm.owner%TYPE
)
IS
BEGIN
    INSERT INTO jcma_farm (farm_name, location, owner) values (ins_farm_name, ins_location, ins_owner);

END;
/
```

Delete chemical

This procedure will take in one parameter(**p_id**) which will be the id number of the chemical, and it will delete the chemical corresponding to that id from our database.

```
CREATE OR REPLACE PROCEDURE delete_chemical
(
     p_id IN jcma_chemicals.chem_id%TYPE
)
as
BEGIN
    Delete
    From jcma_chemicals
    where chem_id=p_id;

End delete_chemical;
/
```

Stored Functions

get_avg(n)

This function will take in one number as its parameter. It will order the weight in the jcma_picks table, by descending order and it will return the average of the n number of weight records.

```
create or replace function get_avg(n in number)
return number
is
  s number(9,2) := 0;
  p number(7,2);
  cursor c is select weight from jcma_picks order by weight DESC;
Begin
    open c;
    for i in 1..n Loop
    fetch c into p;
    s := s+p;
    end Loop;
    close c;
    return s/n;
Exception
    when others then
        raise_application_error(-4001,'An error occured in '||SQLCODE||
            '-Error-' || SQLERRM);
end;
/
```

Stored Trigger

This trigger will be fired when a record of the jcma_farm table is updated or deleted. The trigger will convert the old record and new record as strings and save the old record string and the new record string into a table defined as logTable.

```
CREATE OR REPLACE TRIGGER jcma_farm_log
BEFORE INSERT OR UPDATE OR DELETE ON jcma_farm
FOR EACH ROW
DECLARE
oldrow varchar2(1000);
newrow varchar2(1000);
BEGIN
    IF INSERTING THEN
        oldrow := NULL;
        newrow := TO_CHAR(:new.farm_name)||' '||
            TO_CHAR(:new.location)||' '||
            TO_CHAR(:new.owner);
    ELSIF UPDATING THEN
        oldrow := TO_CHAR(:old.farm_name)||' '||
            TO_CHAR(:old.location)||' '||
            TO_CHAR(:old.owner);
        newrow := TO_CHAR(:new.farm_name)||' '||
            TO_CHAR(:new.location)||' '||
            TO_CHAR(:new.owner);
    ELSIF DELETING THEN
        oldrow := TO_CHAR(:old.farm_name)||' '||
            TO_CHAR(:old.location)||' '||
            TO_CHAR(:old.owner);
        newrow := NULL;
    END IF;
    IF LENGTH(oldrow) > 40 THEN
        oldrow := SUBSTR(oldrow, 0, 39);
    END IF;
    IF LENGTH(newrow) > 40 THEN
        newrow := SUBSTR(newrow, 0, 39);
    END IF;
    INSERT INTO jcma_logtable (oldVal,newVal) VALUES (oldrow, newrow);
END;
/
```

Phase 5

7.1 Description of Daily activities

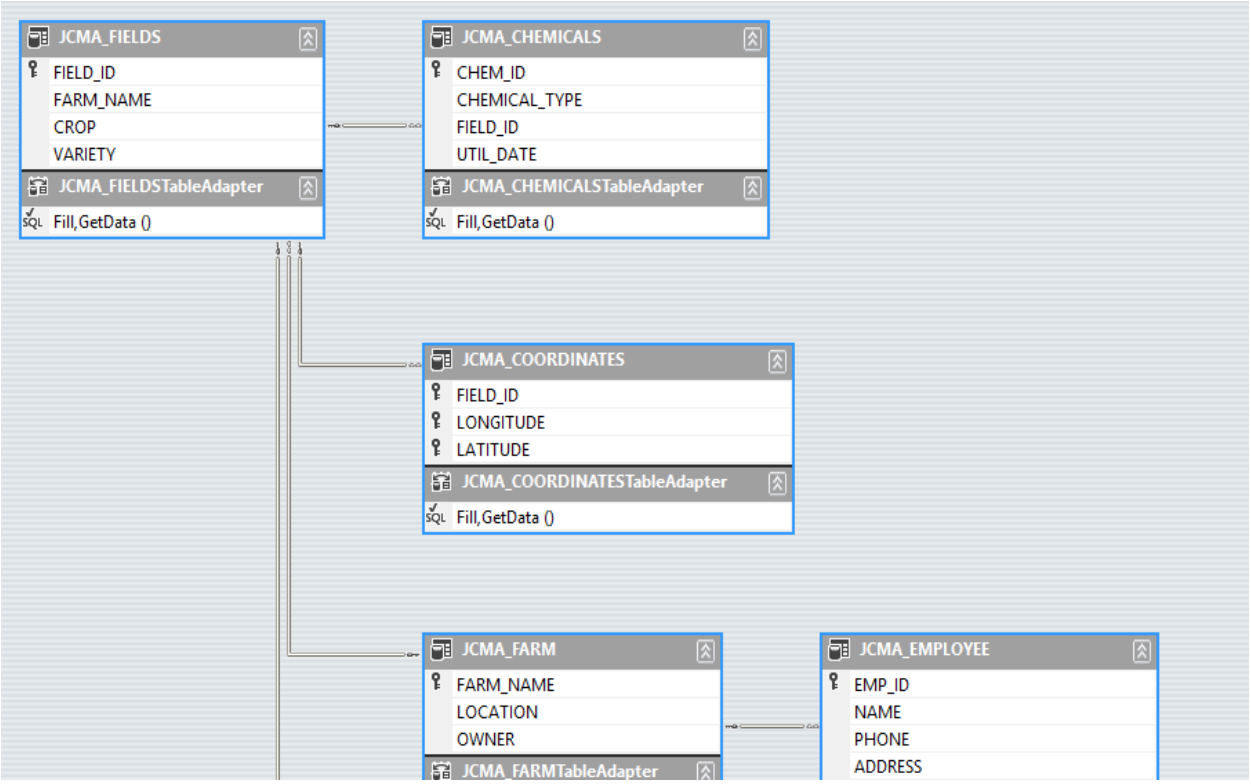
_____The orchard database is meant for the field owners only. The field owner will be able to analyze the performance of the employees hired by looking at the reports of

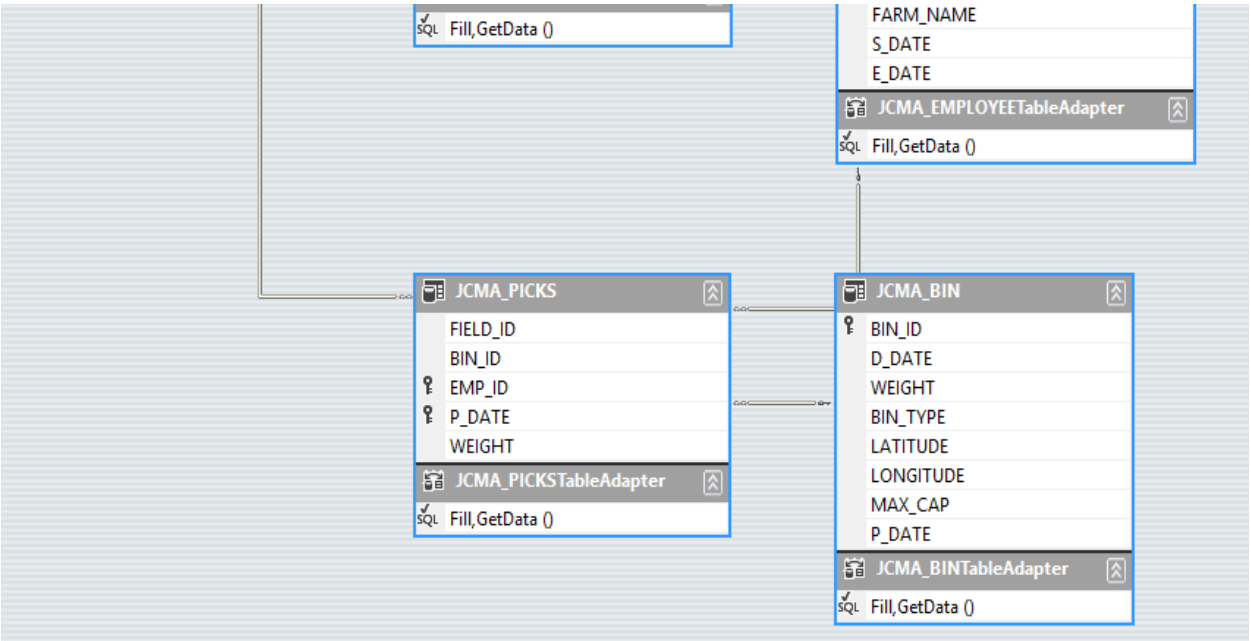
employee performance and employee contribution generated by our database. Farm owners will also be able to keep track of all their fields performance. They will be able to accomplish this by pulling up monthly, and/or yearly reports for the amount of crop collected for their fields. We have also integrated two date/time pickers, which are used as parameters in reports, in order to view data between the dates selected.

Besides generating reports for fields and employees, the farm owner will have access to all his field and employee records. Therefore, he has the ability to add, update and delete records from the database.

7.2 Relation

Our final relational model contains 7 tables. We ended up using 2 procedures which are the insert and the delete procedures. We used a “Before” delete trigger for our “Fields” table, because it contains a parent key that is used by other tables, therefore we had to make sure the foreign key was deleted before the parent key.


JCMA_FIELDS
FIELD_ID
FARM_NAME
CROP
VARIETY
JCMA_FIELDSTableAdapter
Fill,GetData ()
JCMA_CHEMICALS
CHEM_ID
CHEMICAL_TYPE
FIELD_ID
UTIL_DATE
JCMA_CHEMICALSTableAdapter
Fill,GetData ()
JCMA_COORDINATES
FIELD_ID
LONGITUDE
LATITUDE
JCMA_COORDINATESTableAdapter
Fill,GetData ()
JCMA_FARM
FARM_NAME
LOCATION
OWNER
JCMA_FARMTableAdapter
JCMA_EMPLOYEE
EMP_ID
NAME
PHONE
ADDRESS



The following trigger implemented in our database will take effect before any row from the “Fields” table is deleted. We made a before trigger on this table because it contains the Field_ID column which happens to be a parent key for the Picks,Coordinates and Chemicals table.

```
CREATE OR REPLACE TRIGGER del_fields_bypk
    BEFORE DELETE ON JCMA_FIELDS
    FOR EACH ROW
BEGIN
    DELETE FROM JCMA_PICKS
    WHERE JCMA_PICKS.FIELD_ID = :old.FIELD_ID;
    DELETE FROM JCMA_COORDINATES
    WHERE JCMA_COORDINATES.FIELD_ID = :old.FIELD_ID;
    DELETE FROM JCMA_CHEMICALS
    WHERE JCMA_CHEMICALS.FIELD_ID = :old.FIELD_ID;
END;
/
```

-

The following trigger takes place when the user inserts a new field into the database. It will take the value passed in as the field_id which happens to be the primary key and it will replace the value with the next number in our sequence

generator. The reason why we implemented this was because we didn't want the user to insert a repeated field_id otherwise we would have repeated primary keys.

```
CREATE TRIGGER field_id_increment
before insert on jcma_fields
for each row
begin
    select field_id_seq.NEXTVAL into :new.field_id from dual;
end;
/
```

This triggered is executed when the user inserts a new employee into our database. The functionality to this trigger is the same as the field insert trigger, because it will take the emp_id entered by the user and replace it with the next number in our sequence generator that way there are no repeated primary keys.

```
CREATE TRIGGER employee_id_increment
before insert on jcma_employee
for each row
begin
    select emp_id_seq.NEXTVAL into :new.emp_id from dual;
end;
/
```

_____The following procedure is used when the user inserts a new employee into the fields table. This procedure takes in four parameters and will insert the values into a new field row.

```
CREATE OR REPLACE PROCEDURE insert_fields (
    ins_field_id   in jcma_fields.field_id%TYPE,
    ins_farm_name  in jcma_fields.farm_name%TYPE,
    ins_crop       in jcma_fields.crop%TYPE,
    ins_variety    in jcma_fields.variety%TYPE
)
IS
BEGIN
    INSERT INTO jcma_fields (field_id, farm_name, crop, variety) values (ins_field_id, ins_farm_name, ins_crop, ins_variety);
END;
/
```

The following procedure that we use in our front end is the “insert_farm” procedure. This procedure takes in three parameters which will be our columns for the new added row to the farm table.

```
CREATE OR REPLACE PROCEDURE insert_farm (
    ins_farm_name in jcma_farm.farm_name%TYPE,
    ins_location  in jcma_farm.location%TYPE,
    ins_owner     in jcma_farm.owner%TYPE
)
IS
BEGIN
    INSERT INTO jcma_farm (farm_name, location, owner) values (ins_farm_name, ins_location, ins_owner);

END;
/
```

The last procedure that we use in our front end is called whenever the user inserts a new employee. This procedure will take 7 parameters and insert them as the values of the new employee row that is being added.

```
CREATE OR REPLACE PROCEDURE insert_employee (
    ins_emp_id  in jcma_employee.emp_id%TYPE,
    ins_name    in jcma_employee.name%TYPE,
    ins_phone   in jcma_employee.phone%TYPE,
    ins_address in jcma_employee.address%TYPE,
    ins_farm_name  in jcma_employee.farm_name%TYPE,
    ins_s_date  in jcma_employee.s_date%TYPE,
    ins_e_date  in jcma_employee.e_date%TYPE
)
IS
BEGIN
    INSERT INTO jcma_employee (emp_id, name, phone, address, farm_name, s_date, e_date)
    values (ins_emp_id, ins_name, ins_address, ins_phone, ins_farm_name, ins_s_date, ins_e_date)
END;
/
```

7.3 Screenshots

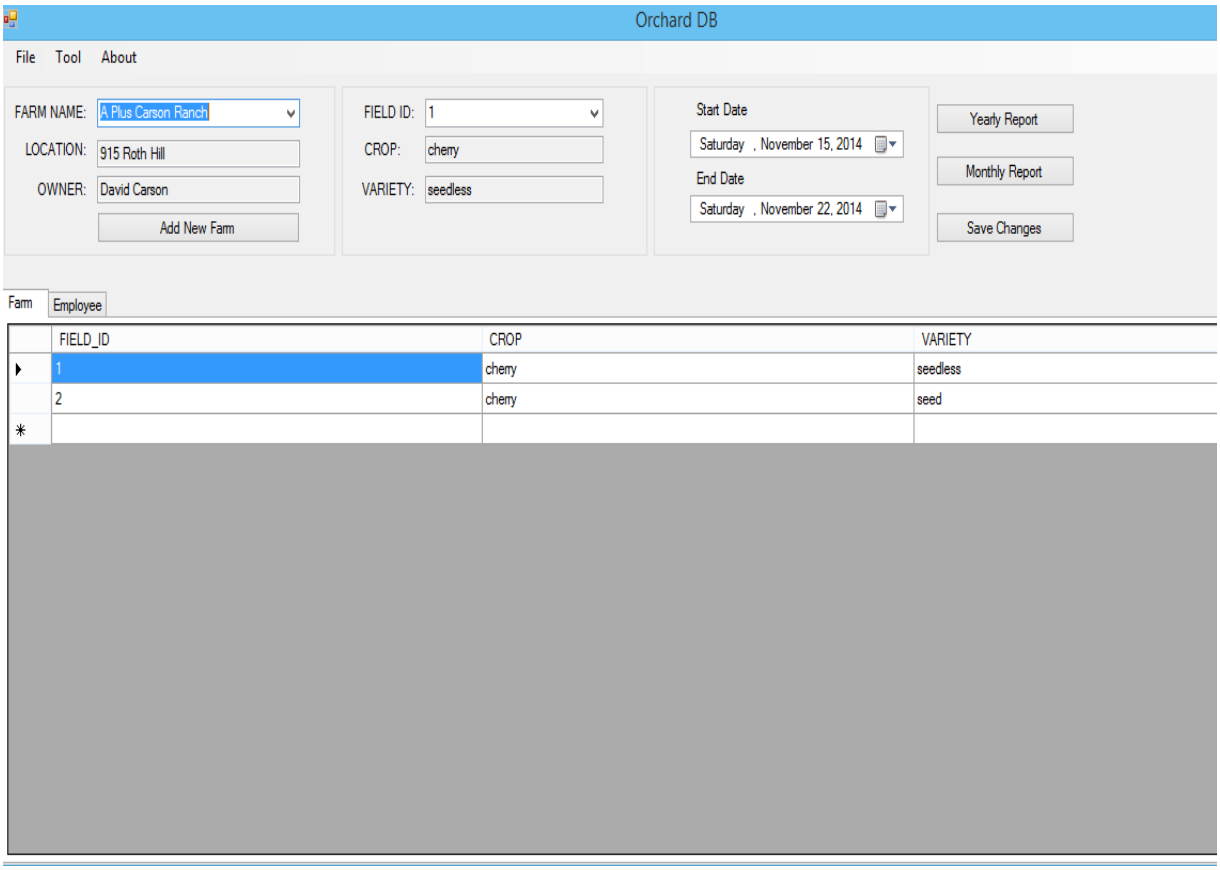
Main Form: This is the form that will show as soon as our program gets executed. We display the farm data in our data grid view, which shows the field_ids, crop, and variety. We included a tab in our UI which is the Employee tab. By clicking on the tab the user will be able to view all the employees that work, and have worked for the farm that has been selected. The employee tab will show the employee ID, name, address, phone number, the starting date for when they first got hired, and the end date which is the date they left, however, if the value is null it means the employee is still working for the farm.

We give the option for the user to add a new farm by clicking on the “Add New Farm” button, which will call another form, where the user will enter the farm. We also have three buttons on the far right side of the form. Two of the buttons will generate a report, one of them is a yearly report, and the other is a monthly report. The “save changes” button will save the changes made on the datagridview.

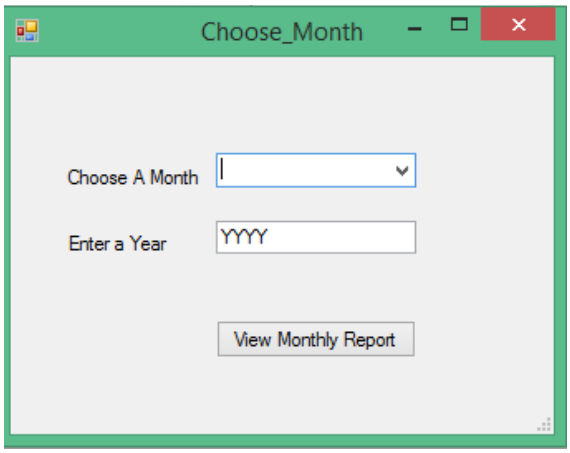
We also allow the user to view the all the picks in a particular field by right clicking any of the rows in the farm table. By right clicking the gridview allow the user to

choose from 2 options. One of those are generating a report and the other one is viewing the picks related to the selected field.

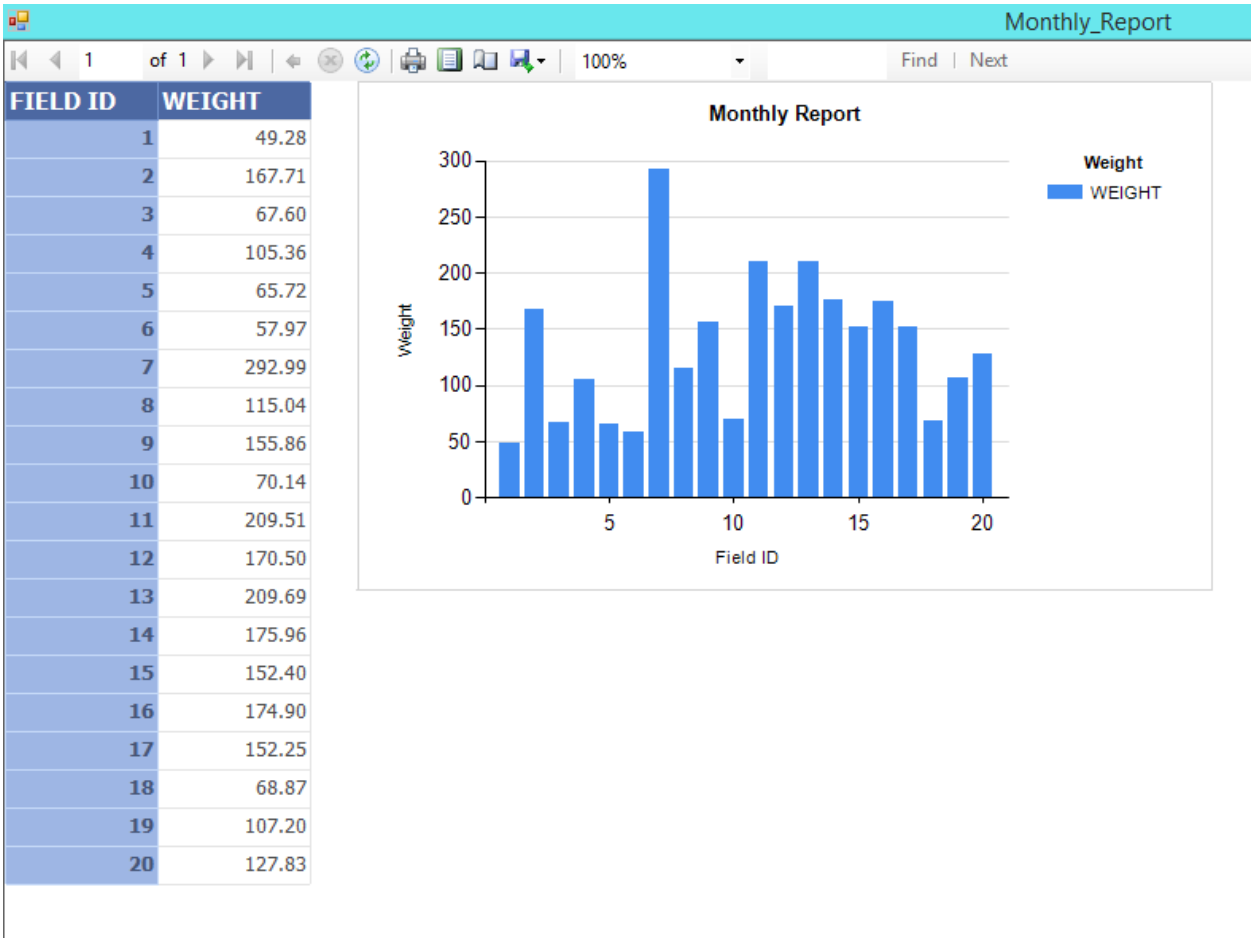
By right clicking on the datagrid view of the Employee tab, the user has a choice of viewing either the employee performance, or the Employee Picks report. By choosing the view this employees picks option, the user can view the picks the employee has had throughout his career.



Choose A Month Form: This form will show after the user clicks the “Monthly Report” button. The user will choose a month and also input the year in a (YYYY) format. After filling up the requirements, the user will click the “View Monthly Report” and the monthly report will be generated.

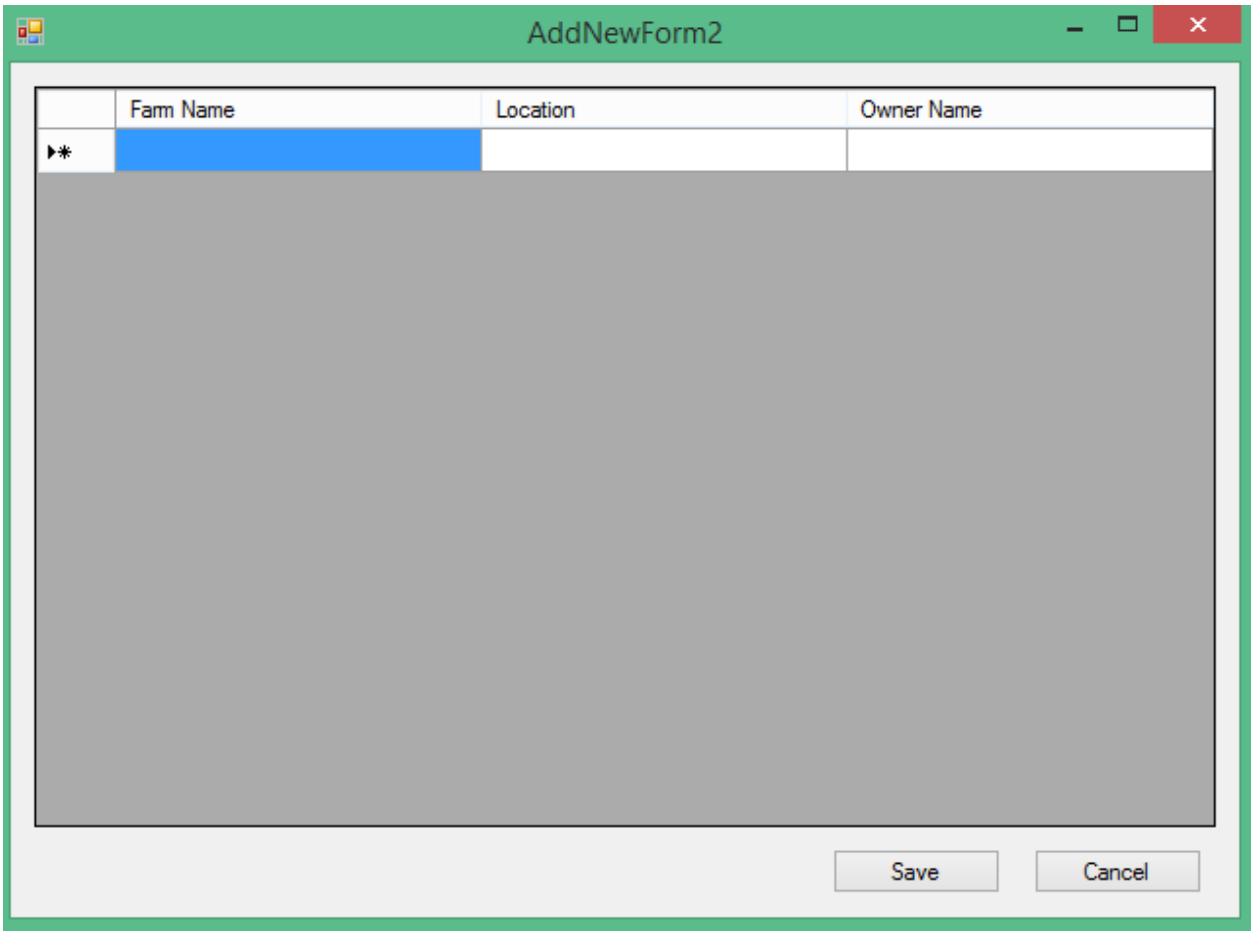


Monthly Report: Once the user clicks on the “View Monthly Report” button the following report will appear. The report shows us a table which has a Field_ID and a Weight column. The Field_ID column shows all the IDs and the Weight column will show the total weight picked on that particular field. We also provided a bar graph shows the weight on the Y-Axis and the field ID in the X-axis. The user has the option to save the report as a PDF, Excel, or Word document by clicking on the “Save” function. The user is also able to print the report by clicking the print icon.



FIELD ID	WEIGHT
1	49.28
2	167.71
3	67.60
4	105.36
5	65.72
6	57.97
7	292.99
8	115.04
9	155.86
10	70.14
11	209.51
12	170.50
13	209.69
14	175.96
15	152.40
16	174.90
17	152.25
18	68.87
19	107.20
20	127.83

Add A Farm Form: The following form will be show when the user clicks the “Add New Farm” button. The user will add as many new farms as he wishes and once he is done editing/adding he/she will click the save button to send the new farms to the database. Before the program inserts the new added farms to the database, it will first check to see if any of the farm names in the form are repeated with the ones in the database. If it finds a repeated farm name it will not insert it in the database because the farm name is our primary key, therefore there can not be any repeated primary keys.



7.4 Major steps of designing a user interface

When designing a user interface, it is important to keep the user in mind. The developer often has extreme familiarity due to the fact that he created it; however, the user will not have the same familiarity. The developer will often be satisfied by just having the functionality in the program and do not always take into account how the user will use the program. Some procedure may not be intuitive to the user but it is functionality complete in the programs. User will not like to do complicated procedure to do a task if it can be simplified. It is important to consider how the user will use the

program and the typical operation of the program. This is the most important point of user interface design.

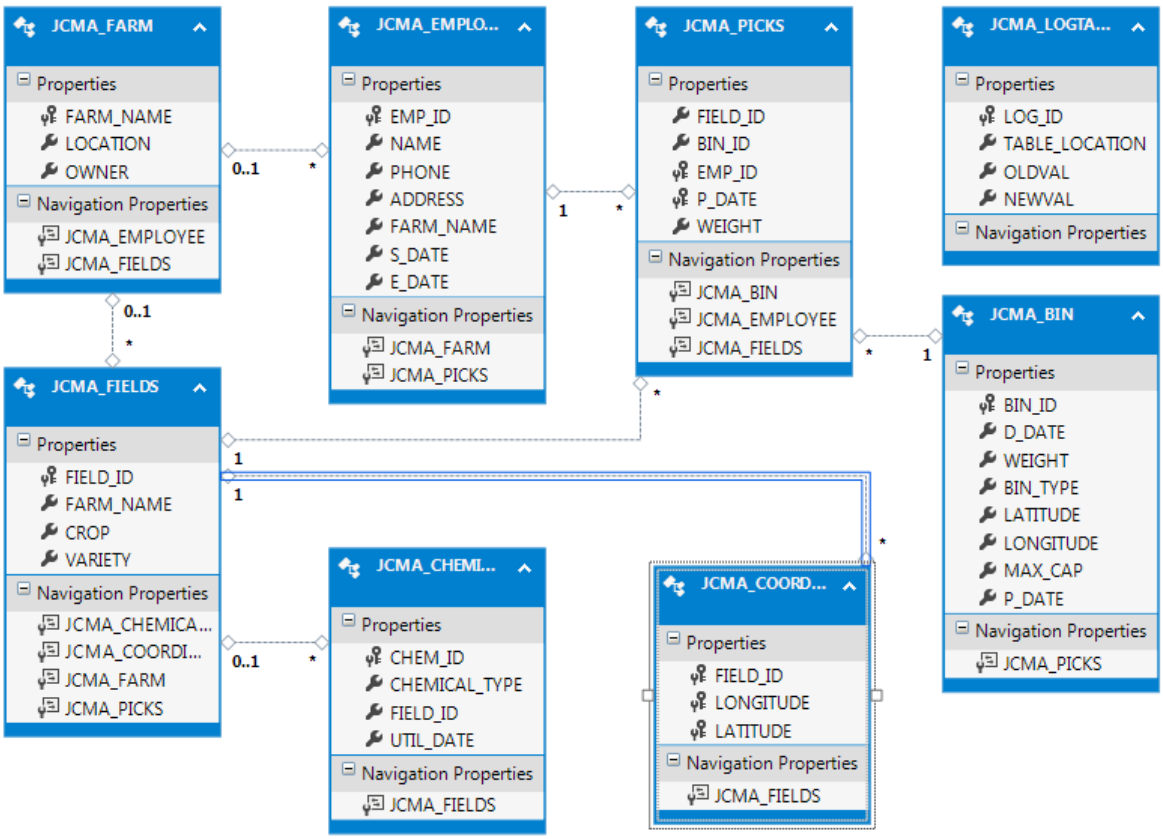
Designing the user interface can be complicated and difficult to predict. Some very large companies have a difficult time designing good user interfaces. The best way to improve the user interface is to have people test the program to get feedback. The best feedback come from users who have little familiarity with the programs and would be using the programs on a consistent basis. These users can provide valuable feedback to improve the program. Another good feedback is to grab people who have no familiarity with the program and get their feedback. Another tester for the program is to force the developer to use their own program consistently for a period of time. If the developer is forced to use the program daily, then he can see the parts of the user interface that can be improved.

Prototyping is a popular tool to give the users an approximation of how the project will look and the behavior can be planned. For example, Microsoft Sketchpad is a prototype tool that allows scripting of of mock up to simulate usage. This will allow the user to get feedback before building the main program.

7.5 Descriptions of major classes

The Gui was very reliant on the use of entity framework's entity data model and data binding. Entity framework is an object relational mapping framework that create a representation of the database called the Entity data model. The Entity data model is an extension of the entity relation model and allows the developer to develop application without referring to the database.

Entity framework has many powerful features and it can handle the create, replace, drop, and updates procedures to the database. It can be used as general connection to the main database and create classes that can be binded to various controls within winform. The changes to the bound object will also be reflected on the controls. Entity framework automatically generates classes based on Entity data model. Important classes generate by Entity framework include JCMA_FARM, JCMA_FIELDS, JCMA_CHEMICALS, JCMA_COORDINATES, JCMA_EMPLOYEE, JCMA_PICKS, and JCMA_BIN. The Entity data Model is shown below.



Each relation classes contains the information from one row in the corresponding relation. For example, an instance of the JCMA_FARM class will contain the data from one row in the relations. Using entity framework and Linq, you can query the database

and return dataset to the programs. The main forms are mainly used to display the data and manipulate the form. Data binding is the main device to transfer data to the forms and the second device used was queries. All grids, text box, and combo boxes uses one of these two devices. An example of using entity frameworks is given below.

```
protected override void OnLoad(EventArgs e)
{
    base.OnLoad(e);
    // create a new context of the entity Framework
    // context allow the program to interact with the database
    _contextEntities = new ODBEntities();
    // Load the data from the database into the context
    _contextEntities.JCMA_FARM.Load();
    _contextEntities.JCMA_FIELDS.Load();
    _contextEntities.JCMA_PICKS.Load();
    // set the binding source to the binding list of JCMA_FARM
    this.jCMA_FARMBindingSource.DataSource = _contextEntities.JCMA_FARM.Local.ToBindingList();
    StartDatePicker.Value = _sDate;
    EndDatePicker.Value = _eDate;
}
```

7.6 Learning a new development tool in a new language

Using a new development tool in a new language did present various difficulties. Every language has various differences that need to be learned to effectively program in a new language. For example, both Java and C# have a concept of an enhanced for loop / foreach; however, they have different syntax and implementation. The basic concepts of programming apply across languages but it can be frustrating learning how to relearn the syntax. In addition, Visual Studio controls have a lot of properties that can be manipulated and it is difficult to achieve the desired effect without understanding the system completely.

Entity Framework was difficult to learn because of the lack of understanding of how it works. It was difficult to bind the data properly and manipulate the data set within the program.

7.7 Major Steps of Designing and Implementing a Database

Designing and implementing an end-to-end database is not an easy task. It takes a lot of effort, research, and patience. Keep in mind, many times throughout the project there will be a lot of refining and redesigning.

The first step a developer should take when it comes to designing a database is to research the company, organization, and business for whom the database is being designed for. One should consider talking to the employees, CEOs, and any other staff that will be using this database that way, one can define the requirements that the database must meet, because it is harder to change them later on.

After we have done our intense research, we are now ready to take all of the information gathered and design and create our E-R(Entity-Relationship) model. An E-R model is a basic model that is intended for managers and other non-technical persons and therefore it can easily be explained to them to make sure your database has covered all of the requirements needed. This step is very important because it plays a big role when it comes to designing the relational model.

Once we have established a sturdy E-R model we are ready to convert it into a relational model. Many DBMS are based on relational models. This process requires converting our entities into relations and having primary and foreign keys inside our relations in order to have access to other relations. It is really important to carefully design the relations in our relational model because the model can be directly applied to our database.

After we have finished designing our relational model, we are now ready to write the subprograms and the triggers that will be used in our database.

The last step on developing a database is the software interface. We must make sure that the GUI is simple enough that way the users won't have a hard time when they utilize our program. Having a good looking GUI plays a huge role, not only does it make it more appealing to the user's eye but it also makes it easier to understand and the functionality of our program will be clear to the user.

It is very important to follow and revise each and every step of designing a database because we must try to fix any errors before moving on. Sometimes a small error at a certain step may not seem critical, but it might come and haunt us later on in other steps. There was many times where we had to go back and refine our E-R and relational models, because we had to fix errors. Encountering these errors and fixing them served us a major learning experience and also served as a lesson to avoid similar errors in any of our future database designs.

Survey

Outcome	Jason	Moises
(3b) An ability to analyze a problem, and identify and define the computing requirements and specifications appropriate to its solution	**10**	**10**
(3e) An ability to design, implement and evaluate a computer-based system, process, component, or program to meet desired needs.	**10**	**9**

An ability to understand the analysis, design, and implementation of a computerized solution to a real-life problem.		
(3f) An ability to communicate effectively with a range of audiences. An ability to write a technical document such as a software specification white paper or a user manual.	**7**	**7**
(3j) An ability to apply mathematical foundations algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.	**9**	**9**